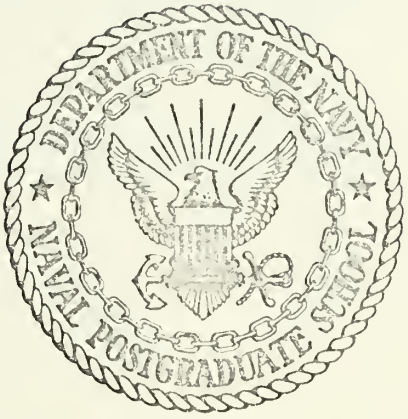
ALGOL-W (SUBSET) COMPILER

Carl Clifford Brewer

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

ALGOL-W (Subset) Compiler

by

Carl Clifford Brewer

Thesis Advisor: Alan B. Roberts

June 1972

ALGOL-W (Subset) Compiler

by

Carl Clifford Brewer
Lieutenant, United States, Navy
B.E.E., Auburn University, 1964

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1972

ABSTRACT

This paper describes the ALGOL-W (subset) compiler and language. The primary objective of this project was to develop a compiler that would accept a modified integer version of ALGOL-W that could be used as a programming system to aid in the study of the compilation and execution of ALGOL-like languages. Included is a brief discussion of the ALGOL-E machine options that complement the compiler's capabilities. This compiler was written in the XPL language.

TABLE OF CONTENTS

I. INTRODUCTION

The programming languages course at the Naval Postgraduate School currently makes use of the language ALGOL-E [Ref. 1], an integer-only ALGOL-like language which has many desirable instructional features. In the latter part of the course the language ALGOL-W (revised) [Ref. 2] is used to investigate some of the more sophisticated programming language features, such as list processing and character string manipulation. The language ALGOL-W (revised) is then used by students in a number of courses which follow in the curricula. The two languages, ALGOL-E and ALGOL-W (revised) are quite similar; however they do have a number of syntactic differences that make the mid-course transition more difficult and time-consuming than is desirable.

The advantage of the ALGOL-E language is in the instructional features included in its implementation; hence the obvious solution to the transition problem seemed to be the implementation of a form of ALGOL-W (revised) with the inclusion of these same teaching aids. A significant portion of the advantage gained in the ALGOL-E programming system is due to features included in the ALGOL-E machine, a simulated stack processing machine. In view of these facts, the decision was made to implement an integer-only subset of ALGOL-W (revised) which would be compatible with the ALGOL-E machine. It was further decided that care would be taken

throughout the project to produce a proper subset of ALGOL-W (revised) so that programs written under the instructional system could also be run on the ALGOL-W compiler without change. Thus, the process of transition from the instructional system to the full ALGOL-W language would be purely one of expansion.

The ALGOL-E programming system was implemented under the XPL compiler generator system discussed in detail by McKeeman, Horning and Wortman [Ref. 3]. The high degree of similarity that exists between ALGOL-E and ALGOL-W and the fact that the ALGOL-E machine was to be used as the object machine of the compilation process made it very desirable to use the XPL system in the generation of the ALGOL-W (subset) compiler. The ALGOL-E compiler served as a model for the development of this compiler.

The objective of this project was to produce the ALGOL-W (subset) compiler to execute on the ALGOL-E machine and to provide the basic material so that the proper instructional documentation may be produced if it is desired that this system be implemented for student access.

II. THE LANGUAGE

The ALGOL-W (subset) language is, as previously mentioned, a proper subset and an integer version of ALGOL-W (revised). Records and references have not been included nor have any string manipulation capabilities.

The syntax provides for: normal integer arithmetic, including exponentiation; ASSIGNMENT statements; GOTO statements;

WHILE statements; FOR statements; IF statements; READ statements; and WRITE statements.

Procedures, both function and proper, are included and have a recursive capability as evidenced in the sample program in Appendix C. All procedure parameters are "call by value;" in fact, the words "INTEGER VALUE" are required in the parameter list in keeping with the ALGOL-W conventions.

The syntax provides indirectly for the null statement that appears in ALGOL-W (revised). This capability is provided within the block head and block end structures. The complete syntax is listed in Appendix A.

Although a GOTO statement is included, it is not implemented in the traditional manner. It can only be used in one direction. The label for the GOTO statement has to occur before the GOTO statement itself in the program. The reason for implementing it in this manner was to avoid the teaching problem that could arise when trying to teach students a block structured language if the only languages they have previously known were FORTRAN or FORTRAN-like. If the GOTO statement had been implemented in the usual manner it would be easy for students to rely on this capability and never learn the block structure programming advantages. Implementing the GOTO statement in the backward direction only makes it impossible to avoid the use of the block structured facility.

All other portions of the language have been implemented in the traditional manner.

III. THE COMPILER

The compiler was written using portions of McKeeman's proto-compiler and Kildall's ALGOL-E compiler as guides. The Backus-Naur Form (BNF) syntax productions were input to McKeeman's syntax analyzing program ANALYZER which produced the parsing tables needed in the compiler. The compiler was written in XPL. The compiler program listing is shown in Appendix B.

A. IMPLEMENTATION

In order to implement the compiler, certain minor changes would have to be made to the ALGOL-E machine and to the ALGOL-W (subset) compiler.

1. Operator REM

The operator REM in ALGOL-W signifies the remainder after integer division. This operator does not exist in the ALGOL-E machine as it is not used in the ALGOL-E language. It could have been implemented in the compiler instead, and initially it was, but it would not be in keeping with the normal scheme for emitting assembly code in the compiler because it is a terminal operand that should have a counterpart in the associated assembly language. This operator could easily be added to the ALGOL-E machine. The only other change needed for this operator would be to initialize the value of the operator and add a call to the code emitting procedure from the approproate case statement in SYNTHESIZE.

2. READ and READON

In ALGOL-W (revised) the READ command always begins with the next data; while the command READON is used to read further from the same card. The READ command, as it is implemented in the ALGOL-E machine, is RDV which reads free-form and a change would be required in the machine in order to properly implement READ and READON in the desired fashion.

B. ADVANTAGES

As previously stated, the main advantage of having this compiler available for the programming languages course is that it is an easy transition from this language to ALGOL-W (revised).

There are other advantages that exist in this compiler, along with its machine, that also exist in the ALGOL-E system.

1. Instructional Capabilities

This system would provide the user with a means of investigating the data structures of one particular system that would enable him to gain insight into the internal workings of compiler systems in general.

2. Compiler and Machine Options

There are compiler options available that allow the user to trace the parsing and code generation processes at compilation-time and machine options that allow traces of execution and dumps of memory at execution-time.

The output formats of traces and dumps are in such a form as to be easily interpreted by the user. All of the options output the information in mnemonic form that relates to the ALGOL-E assembly language.

3. Backus-Naur Form (BNF)

The entire language is formally defined in the Backus-Naur Form (BNF). The tree-like structure of the BNF makes the syntax easy to teach and understand.

IV. TESTING

The testing of the compiler was accomplished in three separate phases. The first phase of testing was the testing of the completed BNF. The second phase was the testing that was accomplished during the actual writing of the compiler program. The third phase consisted of running test programs on the completed compiler.

A. TESTING THE BNF

After the proposed BNF had been successfully run in the syntax analyzing program ANALYZE and the produced tables had been included in the framework of the compiler, null case statements were inserted into the code synthesizing procedure SYNTHESIZE, which is a case statement on the production number, in order to be able to run sample programs that would just exercise the parsing routines. These null statements were to later become the code emitting portion of the compiler.

A number of sample programs were run in this manner and they demonstrated that the compiler would accept programs written in the desired syntax.

B. TESTING DURING PROGRAMMING

The code generation routines were written in a modular fashion based on groups of BNF productions which were closely

related. As each such group was completed, tests were made by running programs that relied on those productions. Errors that became evident during this phase of testing were corrected before proceeding to program further productions. This modular type of programming and testing made the creation and debugging of the compiler significantly easier.

c. FINAL TESTING

After the compiler was completed it was necessary to run programs that would exercise the entire system. These programs would have to test for conflicts between production groups and also check for conflicts resulting from the embedding of structures tested individually. Two such programs were run and both executed properly in the completed compiler. An example of one of these programs is shown in Appendix C.

Time did not permit further testing. As would be the case with any new compiler, adequate testing would have to include running test programs written by a number of users or user groups prior to utilizing it in an instructional capacity.

V. RECOMMENDATIONS

There are tasks that would have to be preformed if the ALGOL-W (subset) compiler is to become the active instructional compiler for the programming languages course in addition to the further testing previously mentioned.

A. DOCUMENTATION

Instructional documentation should be prepared using the ALGOL-E documentation as a basic guideline.

B. EXTENSIONS

Additional ALGOL-W features could be added to the language if desired, and as desired, with comparative ease. Due to the modularity of the compiler structure and the cookbook straightforwardness of the XPL compiler generating system, the process of modifying the language is made easy.

VI. CONCLUSIONS

It has to be concluded that automatic compiler generation is a worthwhile endeavor, as evidenced by the fact that is was possible to produce an operational compiler in the short time alloted for this project. In the past, the creation of a compiler of this size generally took much greater time and effort.

It is felt that the compiler produced during this project could be successfully utilized in the programming languages course. The accomplishments thus far leave a fairly small amount of work to complete the task of providing a complete system ready for use in the classroom.

The writing of the instructional manual would even be a relatively minor task in view of the available ALGOL-E documentation that could be used as a model due to the inherent similarities of the two systems.

GRAMMAR FOR ALGOL-W (SUBSET)
BACKUS-NAUR FORM (BNF)

```
<PROGRAM> ::= <BLOCK> .

<BLOCK> ::= <BLOCK HEAD> <BLOCK END>

<BLOCK HEAD> ::= <PROLOGUE>
     | <BLOCK HEAD> <LABEL IDENTIFIER> <STATEMENT> ;
     | <BLOCK HEAD> <LABEL IDENTIFIER> ;
     | <BLOCK HEAD> <STATEMENT> ;
     | <BLOCK HEAD> ;

<PROLOGUE> ::= <BEGIN>
     | <BLOCK HEAD> <DECLARATION SET> ;

<BEGIN> ::= BEGIN

<DECLARATION SET> ::= <DECLARATION>

<DECLARATION> ::= <SIMPLE VARIABLE DECLARATION>
     | <PROCEDURE DECLARATION>
     | <ARRAY DECLARATION>

<ARRAY DECLARATION> ::= INTEGER ARRAY <ADENTIFIER> <BOUND PAIR LIST>

<BOUND PAIR LIST> ::= <BOUND PAIR HEAD> <BOUND PAIR> )

<BOUND PAIR> ::= <EXPRESSION> : : <EXPRESSION>

<BOUND PAIR HEAD> ::= <BOUND PAIR HEAD> <BOUND PAIR> ,
     | (

<SIMPLE VARIABLE DECLARATION> ::= <TYPE HEAD> <IDENTIFIER>

<TYPE HEAD> ::= <TYPE HEAD> <IDENTIFIER> ,
     | <INTEGER>

<INTEGER> ::= INTEGER

<PROCEDURE DECLARATION> ::= <PROPER PROC DECL>
     | <FUNCTION PROC DECL>

<PROPER PROC DECL> ::= <PROCEDURE> <PROC HEAD> ; <STATEMENT>
```

```
<PROCEDURE> ::= PROCEDURE

<FUNCTION PROC DECL> ::= INTEGER PROCEDURE <PROC HEAD> ; <FUNCTION PROC BODY>

<FUNCTION PROC BODY> ::= <EXPRESSION>
    | <BLOCK HEAD> <EXPRESSION> END

<PROC HEAD> ::= <IDENTIFIER>
    | <IDENTIFIER> ( <FORMAL PARAMETER LIST> )

<FORMAL PARAMETER LIST> ::= <FORMAL PARAMETER SEGMENT>
    | <FORMAL PARAMETER LIST> ; <FORMAL PARAMETER SEGMENT>

<FORMAL PARAMETER SEGMENT> ::= <FORMAL TYPE> <IDENTIFIER>

<FORMAL TYPE> ::= <FORMAL TYPE> <IDENTIFIER> ,
    | <INTEGER> VALUE

<BLOCK END> ::= <STATEMENT> END
    | <LABEL IDENTIFIER> <STATEMENT> END
    | <LABEL IDENTIFIER> END
    | END

<LABEL IDENTIFIER> ::= <IDENTIFIER> :

<STATEMENT> ::= <SIMPLE STATEMENT>
    | <ITERATIVE STATEMENT>
    | <IF STATEMENT>
    | <CASE STATEMENT>

<CASE STATEMENT> ::= <CASE CLAUSE> <BLOCK>

<CASE CLAUSE> ::= CASE <EXPRESSION> OF

<IF STATEMENT> ::= <IF CLAUSE> <STATEMENT>
    | <IF CLAUSE> <SIMPLE STATEMENT> ELSE <STATEMENT>

<IF CLAUSE> ::= IF <LOGICAL EXPRESSION> THEN

<LOGICAL EXPRESSION> ::= <LOGICAL ELEMENT>
    | <RELATION>

<RELATION> ::= <SIMPLE EXPRESSION> <EQUALITY OPERATOR> <SIMPLE EXPRESSION>
    | <LOGICAL ELEMENT> <EQUALITY OPERATOR> <LOGICAL ELEMENT>
    | <SIMPLE EXPRESSION> <RELATIONAL OPERATOR> <SIMPLE EXPRESSION>

<EQUALITY OPERATOR> ::= =
    | ¬=
```

```
<RELATIONAL OPERATOR> ::= <
     | < =
     | >
     | > =

<LOGICAL ELEMENT> ::= <LOGICAL TERM>
     | <LOGICAL ELEMENT> OR <LOGICAL TERM>

<LOGICAL TERM> ::= <LOGICAL FACTOR>
     | <LOGICAL TERM> AND <LOGICAL FACTOR>

<LOGICAL FACTOR> ::= <LOGICAL PRIMARY>
     | ¬ <LOGICAL PRIMARY>

<LOGICAL PRIMARY> ::= <LOGICAL VALUE>
     | ( <LOGICAL EXPRESSION> )

<LOGICAL VALUE> ::= TRUE
     | FALSE

<ITERATIVE STATEMENT> ::= <FOR CLAUSE> <STATEMENT>
     | <WHILE CLAUSE> <STATEMENT>

<WHILE CLAUSE> ::= WHILE <LOGICAL EXPRESSION> DO

<FOR CLAUSE> ::= <FOR HEAD> <FOR END>

<FOR HEAD> ::= FOR <IDENTIFIER> : =

<FOR END> ::= <FOR LIST> DO
     | <EXPRESSION> <INCREMENT> UNTIL <EXPRESSION> DO
     | <EXPRESSION> UNTIL <EXPRESSION> DO

<INCREMENT> ::= STEP <EXPRESSION>

<FOR LIST> ::= <EXPRESSION>
     | <FOR LIST> , <EXPRESSION>

<SIMPLE STATEMENT> ::= <ASSIGNMENT STATEMENT>
     | <PROCEDURE STATEMENT>
     | <GOTO STATEMENT>
     | <BLOCK>
     | <READ STATEMENT>
     | <WRITE STATEMENT>

<GOTO STATEMENT> ::= GOTO <IDENTIFIER>
     | GO TO <IDENTIFIER>
```

```
<PROCEDURE STATEMENT> ::= <CALL HEADING> <CALL BODY> )
    | <PROCEDURE IDENTIFIER>

<CALL HEADING> ::= <PROCEDURE IDENTIFIER> (
    | <CALL HEADING> <CALL BODY> ,

<CALL BODY> ::= <STATEMENT>
    | <EXPRESSION>

<ASSIGNMENT STATEMENT> ::= <LEFT PART> <EXPRESSION>
    | <LEFT PART> <ASSIGNMENT STATEMENT>

<LEFT PART> ::= <VARIABLE> :=

<VARIABLE> ::= <ARRAY DESIGNATOR>
    | <IDENTIFIER>

<ARRAY DESIGNATOR> ::= <IDENTIFIER> <EXPRESSION LIST> <EXPRESSION> )

<EXPRESSION LIST> ::= (
    | <EXPRESSION LIST> <EXPRESSION> ,

<EXPRESSION> ::= <SIMPLE EXPRESSION>
    | <CASE CLAUSE> <EXPRESSION LIST> <EXPRESSION> )
    | <IF CLAUSE> <EXPRESSION> ELSE <EXPRESSION>

<SIMPLE EXPRESSION> ::= <TERM>
    | + <TERM>
    | - <TERM>
    | <SIMPLE EXPRESSION> + <TERM>
    | <SIMPLE EXPRESSION> - <TERM>

<TERM> ::= <TERM> * <FACTOR>
    | <TERM> DIV <FACTOR>
    | <TERM> REM <FACTOR>
    | <FACTOR>

<FACTOR> ::= <FACTOR> ** <PRIMARY1>
    | <PRIMARY>

<PRIMARY1> ::= <NUMBER>
    | <VARIABLE>

<PRIMARY> ::= <NUMBER>
    | <EXPRESSION1> )
    | <VARIABLE>
    | ABS <PRIMARY>
```

```
    | <FUNCTION PROCEDURE CALL>

<EXPRESSION1> ::= ( <EXPRESSION>

<FUNCTION PROCEDURE CALL> ::= <FUNCTION CALL BODY> <IDENTIFIER> )
    | <FUNCTION IDENTIFIER>

<FUNCTION CALL BODY> ::= <FUNCTION IDENTIFIER> (
    | <FUNCTION CALL BODY> <IDENTIFIER> ,

<READ STATEMENT> ::= <READ HEAD> <VARIABLE> )

<READ HEAD> ::= READ (
    | READON (
    | <READ HEAD> <VARIABLE> ,

<WRITE STATEMENT> ::= <WRITE HEAD> <EXPRESSION> )
    | <WRITE HEAD> <STRING> )

<WRITE HEAD> ::= WRITE (
    | WRITEON (
    | <WRITE HEAD> <EXPRESSION> ,
    | <WRITE HEAD> <STRING> ,
```

```
/*
                ALGOL-W(SUBSET)  COMPILER

  /*  FIRST WE INITIALIZE THE GLOBAL CONSTANTS THAT DEPEND UPON THE INPUT
      GRAMMAR.  THE FOLLOWING CARDS ARE PUNCHED BY THE SYNTAX PRE-PROCESSOR  */

  DECLARE NSY LITERALLY '121', NT LITERALLY '52';
  DECLARE V(NSY) CHARACTER INITIAL ( '<ERROR: TOKEN = 0>', '.', ';', ')', ':',
     ',', '(', '=', '¬', '<', '>', '+', '-', '*', 'OF', 'IF', 'OR', 'DO', 'GO',
     'TO', ':=', '**', '|_', 'END', 'AND', 'FOR', 'DIV', 'REM', 'ABS', 'CASE',
     'ELSE', 'THEN', 'TRUE', 'STEP', 'GOTO', 'READ', 'BEGIN', 'ARRAY', 'VALUE',
     'FALSE', 'WHILE', 'UNTIL', 'WRITE', 'READON', 'INTEGER', 'WRITEON',
     '<NUMBER>', '<STRING>', 'PROCEDURE', '<ADENTIFIER>', '<IDENTIFIER>',
     '<FUNCTION IDENTIFIER>', '<PROCEDURE IDENTIFIER>', '<TERM>', '<BLOCK>',
     '<BEGIN>', '<FACTOR>', '<PROGRAM>', '<INTEGER>', '<FOR END>', '<PRIMARY>',
     '<PROLOGUE>', '<RELATION>', '<FOR HEAD>', '<FOR LIST>', '<VARIABLE>',
     '<PRIMARY1>', '<BLOCK END>', '<STATEMENT>', '<TYPE HEAD>', '<PROCEDURE>',
     '<PROC HEAD>', '<IF CLAUSE>', '<INCREMENT>', '<CALL BODY>', '<LEFT PART>',
     '<READ HEAD>', '<BLOCK HEAD>', '<BOUND PAIR>', '<EXPRESSION>',
     '<FOR CLAUSE>', '<WRITE HEAD>', '<DECLARATION>', '<FORMAL TYPE>',
     '<CASE CLAUSE>', '<EXPRESSION1>', '<IF STATEMENT>', '<LOGICAL TERM>',
     '<WHILE CLAUSE>', '<CALL HEADING>', '<LOGICAL VALUE>', '<CASE STATEMENT>',
     '<LOGICAL FACTOR>', '<GOTO STATEMENT>', '<READ STATEMENT>',
     '<DECLARATION SET>', '<BOUND PAIR LIST>', '<BOUND PAIR HEAD>',
     '<LOGICAL ELEMENT>', '<LOGICAL PRIMARY>', '<WRITE STATEMENT>',
     '<EXPRESSION LIST>', '<LABEL IDENTIFIER>', '<PROPER PROC DECL>',
     '<SIMPLE STATEMENT>', '<ARRAY DESIGNATOR>', '<ARRAY DECLARATION>',
     '<SIMPLE EXPRESSION>', '<EQUALITY OPERATOR>', '<FUNCTION PROC DECL>',
     '<FUNCTION PROC BODY>', '<LOGICAL EXPRESSION>', '<FUNCTION CALL BODY>',
     '<ITERATIVE STATEMENT>', '<RELATIONAL OPERATOR>', '<PROCEDURE STATEMENT>',
     '<ASSIGNMENT STATEMENT>', '<PROCEDURE DECLARATION>',
     '<FORMAL PARAMETER LIST>', '<FUNCTION PROCEDURE CALL>',
     '<FORMAL PARAMETER SEGMENT>', '<SIMPLE VARIABLE DECLARATION>');
  DECLARE V_INDEX(22) BIT(8) INITIAL ( 1, 14, 22, 29, 36, 43, 44, 46, 48, 49,
     49, 49, 51, 51, 51, 51, 51, 51, 51, 51, 51, 52, 53);
  DECLARE C1(NSY) BIT(106) INITIAL (
     "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 000",
     "(2) 00000 00000 00000 00000 00200 00000 00000 00000 00000 00000 000",
     "(2) 00200 03000 03300 30030 00020 30033 00003 33000 30333 33020 333",
     "(2) 00222 20222 22222 02200 22022 02200 22020 00000 02000 00000 000",
     "(2) 00201 01100 01100 30020 00020 20013 00002 22000 20220 21000 312",
     "(2) 00000 03000 03300 30020 00000 20033 00002 22000 20220 23200 332",
     "(2) 00000 03010 03300 30020 00000 20033 00102 22001 20221 23200 332",
     "(2) 00000 02020 02200 20000 00000 00022 00200 00002 00000 02000 220",
     "(2) 00000 01100 00000 00000 00000 00000 00100 00001 00000 00000 000",
     "(2) 00000 02100 02200 00000 00000 00020 00000 00000 00000 02000 220",
     "(2) 00000 02100 02200 00000 00000 00020 00000 00000 00000 02000 220",
```

```
   "(2) 00222 20111 11102 00200 00020 00000 22020 00000 02000 00000 000",
   "(2) 00000 01010 01100 00000 00000 00010 00100 00001 00000 01000 110",
   "(2) 00200 00000 00000 00000 00000 00000 00000 00000 00000 00000 000",
   "(2) 00200 00000 00000 00000 00000 00000 00000 00000 00000 00000 000",
   "(2) 00010 00000 00000 00100 00000 00000 01000 00000 00000 00000 000",
   "(2) 00000 00000 00000 00000 00000 00000 00000 00000 00000 00000 100",
   "(2) 00220 20000 00000 00000 00020 00000 00000 00000 00000 00000 000",
   "(2) 00000 01000 01100 00000 00000 00010 00000 00000 00000 01000 110",
   "(2) 00220 20000 00000 00000 00020 00000 20000 00000 00000 00000 000",
   "(2) 00220 20000 00000 00000 00020 00000 20000 00000 00000 00000 000",
   "(2) 00200 00000 00000 00000 00000 00000 00000 00000 00000 00000 000",
   "(2) 00110 00000 00000 00000 00000 00000 00000 00000 00000 00000 000",
   "(2) 00222 20222 22222 00200 02020 02200 22020 00000 02000 00000 000",
   "(2) 00220 00000 00000 00000 00000 00000 00000 00000 00000 00000 000",
   "(2) 00200 00000 00000 00000 00000 00000 00000 00000 00000 00000 000";
DECLARE NC1TRIPLES LITERALLY '310';
DECLARE C1TRIPLES(NC1TRIPLES) FIXED INITIAL ( 132614, 132619, 132620, 132623,
   132636, 132637, 132654, 132658, 132659, 263183, 263197, 263218, 263686,
   263691, 263692, 263695, 263708, 263709, 263726, 263730, 263731, 329222,
   329227, 329228, 329231, 329244, 329245, 329262, 329266, 329267, 394758,
   394763, 394764, 394767, 394780, 394781, 394798, 394802, 394803, 525830,
   525835, 525836, 525852, 525870, 525874, 525875, 722438, 722443, 722444,
   722447, 722460, 722461, 722478, 722482, 722483, 787974, 787979, 787980,
   787983, 787996, 787997, 788014, 788018, 788019, 853510, 853515, 853516,
   853519, 853532, 853533, 853550, 853554, 853555, 984582, 984587, 984588,
   984591, 984604, 984605, 984622, 984626, 984627, 1050118, 1050123, 1050124,
   1050140, 1050158, 1050162, 1050163, 1574406, 1574411, 1574412, 1574428,
   1574446, 1574450, 1574451, 1651204, 1705478, 1705483, 1705484, 1705487,
   1705500, 1705501, 1705518, 1705522, 1705523, 1771014, 1771019, 1771020,
   1771023, 1771036, 1771037, 1771054, 1771058, 1771059, 1836550, 1836555,
   1836556, 1836559, 1836572, 1836573, 1836590, 1836594, 1836595, 1902086,
   1902091, 1902092, 1902095, 1902108, 1902109, 1902126, 1902130, 1902131,
   1920782, 1967622, 1967627, 1967628, 1967631, 1967644, 1967645, 1967662,
   1967666, 1967667, 2164230, 2164235, 2164236, 2164239, 2164252, 2164253,
   2164270, 2164274, 2164275, 2622982, 2622987, 2622988, 2622991, 2623004,
   2623005, 2623022, 2623026, 2623027, 2688518, 2688523, 2688524, 2688527,
   2688540, 2688541, 2688558, 2688562, 2688563, 2707217, 2895922, 4130310,
   4130315, 4130316, 4130319, 4130332, 4130333, 4130350, 4130354, 4130355,
   4149025, 4149033, 4195590, 4195595, 4195596, 4195599, 4195612, 4195613,
   4195630, 4195634, 4195635, 4534789, 4653574, 4653579, 4653580, 4653583,
   4653586, 4653593, 4653596, 4653597, 4653602, 4653603, 4653604, 4653608,
   4653610, 4653611, 4653613, 4653614, 4653618, 4653619, 4653620, 4720134,
   4720139, 4720140, 4720143, 4720156, 4720157, 4720174, 4720178, 4720179,
   4738846, 4916742, 4916747, 4916748, 4916751, 4916764, 4916765, 4916782,
   4916786, 4916787, 4997379, 4997381, 5047814, 5047819, 5047820, 5047823,
   5047836, 5047837, 5047854, 5047858, 5047859, 5059076, 5063682, 5063703,
   5066519, 5309958, 5309963, 5309964, 5309967, 5309980, 5309981, 5309998,
   5310002, 5310003, 5328643, 5328645, 5452293, 5834246, 5834251, 5834252,
```

```
   5834255, 5834268, 5834269, 5834286, 5834290, 5834291, 6358534, 6358539,
   6358540, 6358543, 6358556, 6358557, 6358574, 6358578, 6358579, 6377220,
   6620678, 6620683, 6620684, 6620687, 6620700, 6620701, 6620718, 6620722,
   6620723, 6639363, 6639365, 6702082, 6702103, 7079430, 7079435, 7079436,
   7079439, 7079452, 7079453, 7079470, 7079474, 7079475, 7352835, 7352837,
   7472646, 7472651, 7472652, 7472655, 7472668, 7472669, 7472686, 7472690,
   7472691, 7733804);
DECLARE PRTB(141) FIXED INITIAL (0, 54, 5072452, 19814, 19780, 19807, 77,
   3278454, 5530959, 3302735, 24910, 22858, 19521, 20815, 20783, 1647, 28722,
   85, 50, 17714, 21298, 19521, 24910, 25935, 28722, 20815, 20783, 22858, 35,
   43, 51, 42, 45, 52, 0, 0, 1651204, 9, 10, 8, 0, 0, 0, 7503, 1330194767,
   5187919, 10351, 64, 65, 19791, 26180, 68, 102, 0, 3951, 0, 0, 58, 0, 0, 0,
   0, 0, 4627, 69, 83, 34, 0, 0, 0, 0, 27403, 27404, 11, 12, 0, 84, 0, 0,
   13581, 13594, 13595, 0, 0, 63, 28, 0, 0, 0, 0, 0, 14357, 77, 4605698,
   4745246, 72, 80, 88, 0, 5178372, 4738846, 16389, 33, 6, 75, 0, 0, 0, 0, 0,
   25104, 0, 0, 0, 22296, 0, 0, 0, 2893105, 25196, 0, 8, 0, 0, 0, 0, 0, 0,
   27500, 27506, 0, 0, 741361410, 0, 0, 75, 0, 0, 0, 30210, 0, 0);
DECLARE PRDTB(141) BIT(8) INITIAL (0, 1, 4, 5, 6, 9, 7, 32, 107, 103, 16, 92,
   132, 136, 137, 70, 128, 123, 42, 21, 36, 135, 18, 105, 131, 140, 141, 95,
   133, 134, 130, 138, 139, 94, 19, 104, 77, 60, 62, 58, 57, 59, 61, 48, 79,
   80, 75, 78, 100, 30, 39, 38, 40, 41, 51, 71, 10, 37, 72, 23, 120, 122, 27,
   91, 20, 35, 90, 31, 102, 129, 93, 112, 113, 110, 111, 109, 47, 87, 8, 114,
   115, 117, 116, 22, 76, 125, 119, 3, 53, 121, 124, 118, 2, 26, 50, 49, 73,
   74, 96, 17, 108, 83, 81, 127, 98, 29, 82, 97, 11, 45, 64, 63, 69, 46, 66,
   65, 86, 88, 15, 55, 52, 68, 67, 89, 24, 43, 101, 14, 54, 56, 106, 25, 28,
   44, 85, 99, 84, 13, 126, 34, 33, 12);
DECLARE HDTB(141) BIT(8) INITIAL (0, 57, 77, 77, 77, 61, 77, 71, 79, 105, 96,
   115, 94, 100, 100, 99, 119, 60, 102, 69, 83, 76, 97, 101, 112, 81, 81, 89,
   76, 76, 112, 81, 81, 89, 97, 101, 63, 114, 114, 108, 108, 114, 114, 84,
   59, 59, 88, 59, 75, 110, 67, 67, 67, 67, 72, 90, 55, 83, 90, 58, 66, 60,
   70, 93, 121, 120, 93, 71, 65, 119, 115, 107, 107, 107, 107, 107, 91, 104,
   61, 53, 53, 53, 53, 69, 80, 60, 56, 77, 111, 66, 60, 56, 54, 103, 86, 86,
   113, 113, 74, 78, 79, 64, 73, 85, 116, 110, 64, 74, 95, 68, 98, 98, 99,
   68, 87, 87, 104, 104, 106, 62, 111, 92, 92, 104, 117, 68, 65, 82, 62, 62,
   79, 117, 109, 68, 104, 116, 104, 82, 60, 118, 118, 82);
DECLARE PRLENGTH(141) BIT(8) INITIAL (0, 2, 4, 3, 3, 3, 2, 4, 4, 4, 3, 3, 3,
   3, 3, 3, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 1, 1, 4, 2,
   2, 2, 1, 1, 1, 3, 5, 4, 3, 2, 2, 3, 3, 2, 2, 1, 3, 1, 1, 2, 1, 1, 1, 1, 1,
   3, 2, 2, 2, 1, 1, 1, 1, 3, 3, 2, 2, 1, 2, 1, 1, 3, 3, 3, 1, 1, 2, 2, 2, 1, 1,
   1, 1, 1, 3, 2, 4, 4, 2, 2, 2, 1, 4, 4, 3, 2, 2, 2, 1, 1, 1, 1, 1, 3, 1, 1,
   1, 3, 1, 1, 1, 4, 3, 1, 2, 1, 1, 1, 1, 1, 1, 3, 3, 1, 1, 5, 1, 1, 2, 1, 1,
   1, 3, 1, 1);
DECLARE CONTEXT_CASE(141) BIT(8) INITIAL (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0,
   0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
    0, 0, 0, 0, 0);
DECLARE LEFT_CONTEXT(4) BIT(8) INITIAL ( 63, 21, 48, 70, 49);
DECLARE LEFT_INDEX(69) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    1, 1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
    4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
    5, 5, 5, 5, 5, 5, 5, 5);
DECLARE CONTEXT_TRIPLE(0) FIXED INITIAL ( 0);
DECLARE TRIPLE_INDEX(69) BIT(8) INITIAL ( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 1);
DECLARE PR_INDEX(121) BIT(8) INITIAL ( 1, 2, 7, 18, 19, 28, 36, 41, 41, 42,
    43, 43, 43, 43, 44, 44, 44, 48, 48, 48, 49, 49, 49, 54, 54, 54, 54, 54,
    54, 54, 54, 55, 56, 56, 56, 56, 57, 57, 58, 59, 59, 59, 59, 59, 60, 60,
    62, 62, 63, 63, 69, 70, 71, 76, 78, 79, 83, 83, 84, 85, 87, 88, 89, 89,
    89, 91, 92, 93, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 108, 108, 108,
    109, 109, 109, 109, 110, 112, 112, 112, 113, 114, 116, 117, 118, 118, 119,
    119, 121, 123, 124, 124, 124, 125, 126, 127, 128, 131, 131, 132, 133, 133,
    133, 134, 134, 135, 137, 138, 138, 139, 141, 142);

/*  END OF CARDS PUNCHED BY SYNTAX                                          */

DECLARE
    MAXSYM     LITERALLY      '256',          /* MAX SYMBOLS IN TABLE */
    SIMPLE     LITERALLY      '1',            /* SIMPLE VARIABLE */
    MAXPRT     LITERALLY      '127',          /* EXECUTION PRT MAX SIZE */
    DATALENGTH LITERALLY      '899',          /* SIZE OF DATA ARRAY (3600B)*/
    MAXDATA    LITERALLY      '895',          /* MAX DATA & PRT */
    VECTOR     LITERALLY      '2',            /* VECTOR STORAGE */
    CONSTANT   LITERALLY      '3',            /* NUMERIC CONSTANT */
    LABELV     LITERALLY      '5',            /* CODE OR DATA LABEL */
    ARRAY      LITERALLY      '6',
    FUNC       LITERALLY      '7',
    PROC       LITERALLY      '8',
    IFUNC      LITERALLY      '9',            /* INTRINSIC FUNCTION */
    IPROC      LITERALLY      '10',           /* INTRINSIC PROCEDURE */
    OPERATORS  LITERALLY      '49',           /* NUMBER OF OPERATORS */
    STORAGE    LITERALLY      '3',            /* STORAGE OPCODES */
    MAXCODE    LITERALLY      '4095',         /* MAX NUMBER OF WORDS */
    MAXSYL     LITERALLY      '16384',        /* MAXCODE*4    */
    LEFT       LITERALLY      'MP',           /* LEFT END OF PRODUCTION */
    RIGHT      LITERALLY      'SP',           /* RIGHT END OF PRODUCTION */
    IDENTV     LITERALLY      '1',
    MAXLEVEL   LITERALLY      '31',
    DUMMY      LITERALLY      '';
DECLARE
    FSTACK(30) CHARACTER, /* TO DETERMINE <FUNCTION IDENTIFIER> */
    PSTACK(30) CHARACTER, /* TO DETERMINE <PROCEDURE IDENTIFIER> */
```

```
    SYMBOL (MAXSYM) CHARACTER,          /* SYMBOL TABLE */
    TYPE(MAXSYM) BIT(8),               /* "TYPE" OF SYMBOL (SIMPLE, LABEL...) */
    DATA(DATALENGTH) FIXED,             /* FIXED DATA AND PRT (ALSO FILE BUFF)*/
    LOCATION(MAXSYM) FIXED,             /* HOLDS PR, LIST, AND STRING INFO */
    EXSTACK FIXED,            /* SIZE OF EXECUTION STACK (8192 IF STACKALL=1) */
    STACKALL BIT(1),          /* =1 IF ALL ALLOC IN STACK, 0 IF ALLOC IN FSA */
    CODE(MAXCODE) FIXED,                /* HOLDS GENERATED CODE SYLLABLES */
    CODEW FIXED,                        /* CODEW: CODE WORD (TEMPORARY) */
    (PRTCNT,DATACNT) FIXED,
    (OPCODE,STORECODE) CHARACTER,       /* OPERATION AND STORAGE CODES */
    (SY, /* SYMBOL TABLE COUNTER */ SYLCNT, /* SYLLABLE COUNTER */
    PRTMAX,   /* LARGEST PRT CELL NUMBER */
    ARRCNT,                     /* ARRAY SUBSCRIPT COUNTER (AND FORMAL PARMS) */
    I8,I9,I10,I11,
    I12,I13,I14, I15,I16,
    I17,I18,I19,I20,I21,I22,
    I1,I2,I3,I4,I5,I6,I7 /* GLOBAL TEMPORARIES */ ) FIXED;
DECLARE (ADD,MIN,MUL,DIV,EXP,NEG,RND,TRU,LSS,LEQ,EQL,NEQ,GEQ,GTR,NOT,AND,BOR,
    IM1,IM2,LOD,STO,PRO,RTN,XIT,DEL,DUP,XCH,INX,BFN,BBN,BFC,BBC,BRS,BSC,NOP,
    GET,RET,RDV,WRV,WRS,DMP,TAB,SUB,BIF,ROW,SUP,SAV,UNS,STD) BIT(8);
DECLARE REM BIT(8);
DECLARE BLOCK(MAXLEVEL) FIXED, (BLK,SYLEVEL) BIT(8), PRTLEV(MAXLEVEL) BIT(16);
DECLARE(FIDENT,PIDENT,COLON,COLON_EQUALS,SPLAT,SPLAT_SPLAT) BIT(8);
DECLARE(ADENTIFIER) BIT(8);


   /* TOKEN IS THE INDEX INTO THE VOCABULARY V() OF THE LAST SYMBOL SCANNED,
      CP IS THE POINTER TO THE LAST CHARACTER SCANNED IN THE CARDIMAGE,
      BCD IS THE LAST SYMBOL SCANNED (LITERAL CHARACTER STRING). */
   DECLARE (TOKEN, CP) FIXED, BCD CHARACTER;
   DECLARE EJECT_PAGE LITERALLY 'OUTPUT(1) = PAGE',
      PAGE CHARACTER INITIAL ('1'), DOUBLE CHARACTER INITIAL ('0'),
      DOUBLE_SPACE LITERALLY 'OUTPUT(1) = DOUBLE',
      SINGLE CHARACTER INITIAL (' '),SINGLE_SPACE LITERALLY 'OUTPUT(1)=SINGLE',
      X70 CHARACTER INITIAL ('
                    ');

   /* LENGTH OF LONGEST SYMBOL IN V */
   DECLARE (RESERVED_LIMIT, MARGIN_CHOP) FIXED;

      DECLARE FORLIST(30) FIXED;
    DECLARE(S1,S2) CHARACTER;
 /* USED FOR GOTO LABELS */
    DECLARE LABEL_NAME(30) CHARACTER;
    DECLARE LABEL_LOC(30) FIXED;

   /* CHARTYPE() IS USED TO DISTINGUISH CLASSES OF SYMBOLS IN THE SCANNER.
      TX() IS A TABLE USED FOR TRANSLATING FROM ONE CHARACTER SET TO ANOTHER.
```

```
   CONTROL() HOLDS THE VALUE OF THE COMPILER CONTROL TOGGLES SET IN $ CARDS.
   NOT_LETTER_OR_DIGIT() IS SIMILIAR TO CHARTYPE() BUT USED IN SCANNING
   IDENTIFIERS ONLY.

   ALL ARE USED BY THE SCANNER AND CONTROL() IS SET THERE.
*/
DECLARE (CHARTYPE, TX) (255) BIT(8),
        (CONTROL, NOT_LETTER_OR_DIGIT)(255) BIT(1);

/* ALPHABET CONSISTS OF THE SYMBOLS CONSIDERED ALPHABETIC IN BUILDING
   IDENTIFIERS     */
 DECLARE ALPHABET CHARACTER INITIAL('ABCDEFGHIJKLMNOPQRSTUVWXYZ');

/* BUFFER HOLDS THE LATEST CARDIMAGE,
   TEXT HOLDS THE PRESENT STATE OF THE INPUT TEXT
   (NOT INCLUDING THE PORTIONS DELETED BY THE SCANNER),
   TEXT_LIMIT IS A CONVENIENT PLACE TO STORE THE POINTER TO THE END OF TEXT,
   CARD_COUNT IS INCREMENTED BY ONE FOR EVERY SOURCE CARD READ,
   ERROR_COUNT TABULATES THE ERRORS AS THEY ARE DETECTED,
   SEVERE_ERRORS TABULATES THOSE ERRORS OF FATAL SIGNIFICANCE.
*/
DECLARE (BUFFER, TEXT) CHARACTER,
   (TEXT_LIMIT, CARD_COUNT, ERROR_COUNT, SEVERE_ERRORS, PREVIOUS_ERROR) FIXED
   ;

/* NUMBER_VALUE CONTAINS THE NUMERIC VALUE OF THE LAST CONSTANT SCANNED,
*/
DECLARE NUMBER_VALUE FIXED;

/* EACH OF THE FOLLOWING CONTAINS THE INDEX INTO V() OF THE CORRESPONDING
   SYMBOL.   WE ASK:    IF TOKEN = IDENT    ETC.    */
 DECLARE (IDENT, NUMBER, EOFILE, STRING, SEMI, BEGINV, ENDV) FIXED;

/* STOPIT() IS A TABLE OF SYMBOLS WHICH ARE ALLOWED TO TERMINATE THE ERROR
   FLUSH PROCESS.  IN GENERAL THEY ARE SYMBOLS OF SUFFICIENT SYNTACTIC
   HIERARCHY THAT WE EXPECT TO AVOID ATTEMPTING TO START CHECKING AGAIN
   RIGHT INTO ANOTHER ERROR PRODUCING SITUATION.  THE TOKEN STACK IS ALSO
   FLUSHED DOWN TO SOMETHING ACCEPTABLE TO A STOPIT() SYMBOL.
   FAILSOFT IS A BIT WHICH ALLOWS THE COMPILER ONE ATTEMPT AT A GENTLE
   RECOVERY.   THEN IT TAKES A STRONG HAND.   WHEN THERE IS REAL TROUBLE
   COMPILING IS SET TO FALSE, THEREBY TERMINATING THE COMPILATION.
*/
DECLARE STOPIT(100) BIT(1), (FAILSOFT, COMPILING) BIT(1);
DECLARE S CHARACTER;  /* A TEMPORARY USED VARIOUS PLACES */
/* THE ENTRIES IN PRMASK() ARE USED TO SELECT OUT PORTIONS OF CODED
   PRODUCTIONS AND THE STACK TOP FOR COMPARISON IN THE ANALYSIS ALGORITHM */
DECLARE PRMASK(5) FIXED INITIAL (0, 0, "FF", "FFFF", "FFFFFF", "FFFFFFFF");
```

```
/*THE PROPER SUBSTRING OF POINTER IS USED TO PLACE AN  |  UNDER THE POINT
   OF DETECTION OF AN ERROR DURING CHECKING.  IT MARKS THE LAST CHARACTER
   SCANNED.  */
DECLARE POINTER CHARACTER INITIAL ('
                                   |');
DECLARE CALLCOUNT(20) FIXED   /* COUNT THE CALLS OF IMPORTANT PROCEDURES */
   INITIAL(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

/* RECORD THE TIMES OF IMPORTANT POINTS DURING CHECKING */
DECLARE CLOCK(5) FIXED;

/* COMMONLY USED STRINGS */
DECLARE X1 CHARACTER INITIAL(' '), X4 CHARACTER INITIAL('    ');
DECLARE PERIOD CHARACTER INITIAL ('.');

/* TEMPORARIES USED THROUGHOUT THE COMPILER */
DECLARE (I, J, K, L) FIXED;
DECLARE TRUE LITERALLY '1', FALSE LITERALLY '0', FOREVER LITERALLY 'WHILE 1';

/*  THE STACKS DECLARED BELOW ARE USED TO DRIVE THE SYNTACTIC
   ANALYSIS ALGORITHM AND STORE INFORMATION RELEVANT TO THE INTERPRETATION
   OF THE TEXT.  THE STACKS ARE ALL POINTED TO BY THE STACK POINTER SP.  */

DECLARE STACKSIZE LITERALLY '75';  /* SIZE OF STACK  */
DECLARE PARSE_STACK (STACKSIZE) BIT(8); /* TOKENS OF THE PARTIALLY PARSED
                                           TEXT */
DECLARE VAR (STACKSIZE) CHARACTER;/* EBCDIC NAME OF ITEM */
DECLARE FIXV (STACKSIZE) FIXED;   /* FIXED (NUMERIC) VALUE */

/* SP POINTS TO THE RIGHT END OF THE REDUCIBLE STRING IN THE PARSE STACK,
   MP POINTS TO THE LEFT END, AND
   MPP1 = MP+1.
*/
DECLARE (SP, MP, MPP1) FIXED;



/*              P R O C E D U R E S                                  */


PAD:
   PROCEDURE (STRING, WIDTH) CHARACTER;
      DECLARE STRING CHARACTER, (WIDTH, L) FIXED;
      L = LENGTH(STRING);
      IF L >= WIDTH THEN RETURN STRING;
      ELSE RETURN STRING || SUBSTR(X70, 0, WIDTH-L);
```

```
   END PAD;

I_FORMAT:
   PROCEDURE (NUMBER, WIDTH) CHARACTER;
      DECLARE (NUMBER, WIDTH, L) FIXED, STRING CHARACTER;
      STRING = NUMBER;
      L = LENGTH(STRING);
      IF L >= WIDTH THEN RETURN STRING;
      ELSE RETURN SUBSTR(X70, 0, WIDTH-L) || STRING;
   END I_FORMAT;

ERROR:
   PROCEDURE(MSG, SEVERITY);
      /* PRINTS AND ACCOUNTS FOR ALL ERROR MESSAGES */
      /* IF SEVERITY IS NOT SUPPLIED, 0 IS ASSUMED */
      DECLARE MSG CHARACTER, SEVERITY FIXED;
      ERROR_COUNT = ERROR_COUNT + 1;
      /* IF LISTING IS SUPPRESSED, FORCE PRINTING OF THIS LINE */
      IF ¬ CONTROL(BYTE('L')) THEN
         OUTPUT = I_FORMAT (CARD_COUNT, 4) || ' |' || BUFFER || '|';
      OUTPUT = SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP);
      OUTPUT = '*** ERROR, ' || MSG ||
            '.  LAST PREVIOUS ERROR WAS DETECTED ON LINE ' ||
            PREVIOUS_ERROR || '.  ***';
      PREVIOUS_ERROR = CARD_COUNT;
      IF SEVERITY > 0 THEN
         IF SEVERE_ERRORS > 25 THEN
            DO;
               OUTPUT = '*** TOO MANY SEVERE ERRORS, CHECKING ABORTED ***';
               COMPILING = FALSE;
            END;
         ELSE SEVERE_ERRORS = SEVERE_ERRORS + 1;
   END ERROR;

 SEARCH: PROCEDURE (C,D) FIXED;
         DECLARE (C,D) CHARACTER, (I,L) FIXED;
         L=LENGTH(C)-1;
            DO I=0 TO L BY 3;
            IF D=SUBSTR(C,I,3) THEN RETURN I/3;
            END;
         RETURN -1; /* CODE NOT FOUND */
    END SEARCH;
```

```
  /*  CODE EMITTING PROCEDURES */

STUFFBACK: PROCEDURE(REF,LOC) FIXED;
    DECLARE (REF,LOC,T,I) FIXED;
    I=SHR(REF,2);
    T=CODE(I);
    IF CONTROL(BYTE('C')) THEN OUTPUT='         (BACKSTUFF '||REF||','||
    (REF+1)||': '||LOC||')';
        DO CASE (REF &"3");
        /* 0 */
            DO; REF=SHR(T,16); CODE(I)=(T& "FFFF") | SHL(LOC,16);
            END;
        /* 1 */
            DO; REF= "FFFF" & SHR(T,8);
            CODE(I)=(T& "FF0000FF") | SHL(LOC,8);
            END;
        /* 2 */
            DO; REF="FFFF" & T; CODE(I)=(T& "FFFF0000") | LOC;
            END;
        /* 3 */
            DO; REF=SHL((T & "FF"),8) | SHR(CODE(I+1),24);
            CODE(I)=(T&"FFFFFF00") | SHR(LOC,8); T=CODE(I+1);
            CODE(I+1)= SHL((LOC & "FF"),24) | (T & "FFFFFF");
            END;
        END /* OF CASES */ ;
    RETURN REF;
    END STUFFBACK;

WRITECODE: PROCEDURE(C);
    DECLARE (C,D) CHARACTER, I FIXED;
    D=SYLCNT; I=LENGTH(D); D=SUBSTR('              ',0,13-I)||D;
    OUTPUT=D||' | '||C;
    END WRITECODE;

BACKUP: PROCEDURE;
    SYLCNT=SYLCNT-1;
        DO CASE SYLCNT & "3";
        CODEW=0;                CODEW=CODEW & "FF000000";
        CODEW=CODEW& "FFFF0000"; CODEW=CODEW & "FFFFFF00";
        END;
  END BACKUP;

EMIT1: PROCEDURE (S);
    DECLARE (I,S) FIXED;
    IF CONTROL(BYTE('C')) THEN
        DO; STORECODE=S;
            DO WHILE LENGTH(STORECODE)<3; STORECODE='0'||STORECODE;
```

```
            END;
        I=S&"7F";
        IF (S<128)|(I>OPERATORS) THEN STORECODE='    '||STORECODE;  ELSE
        STORECODE=SUBSTR(OPCODE,I*3,3)||' '||STORECODE;
        CALL WRITECODE(STORECODE);
        END;
        DO CASE SYLCNT & "3";
        CODEW = SHL(S,24);
        CODEW= CODEW | SHL(S,16);
        CODEW= CODEW | SHL(S,8);
            DO; CODEW = CODEW | S; CODE(SHR(SYLCNT,2))=CODEW;
            END;
        END;
    SYLCNT=SYLCNT+1;
    IF SYLCNT > MAXSYL THEN
        DO;
        CALL ERROR('CODE TABLE OVERFLOW: '||MAXSYL||' SYLLABLES GENERATED',2);
        CALL EXIT;
        END;
    END EMIT1;

EMIT0: PROCEDURE(O);
    DECLARE O FIXED;
    O = O |"80";
    CALL EMIT1(O);
    END EMIT0;

EMIT2: PROCEDURE (C);
    DECLARE C FIXED;
        CALL EMIT1(SHR(C,8)); CALL EMIT1(SHR(SHL(C,24),24));
    END EMIT2;

EMITLIT: PROCEDURE(S);
    DECLARE S FIXED;
        IF S<128 THEN CALL EMIT1(S); ELSE
        IF S<256 THEN
            DO; CALL EMIT0(IM1); CALL EMIT1(S);
            END; ELSE
        IF S>=65536 THEN
        CALL ERROR('LITERAL '||S||' IS TOO LARGE',2); ELSE
            DO; CALL EMIT0(IM2); CALL EMIT2(S);
            END;
    END EMITLIT;

  /* END OF CODE EMITTING PROCEDURES */

  /*                    CARD IMAGE HANDLING PROCEDURE                    */
```

```
GET_CARD:
   PROCEDURE;
      /* DOES ALL CARD READING AND LISTING                                    */
      DECLARE (I,J) FIXED, REST CHARACTER;
        AGAIN: BUFFER=INPUT;
            IF LENGTH(BUFFER) = 0 THEN
               DO; /* SIGNAL FOR EOF */
                  CALL ERROR ('EOF MISSING OR COMMENT STARTING IN COLUMN 1.',1);
                  BUFFER=PAD(' ;END EOF ',80);
               END;
            ELSE CARD_COUNT = CARD_COUNT + 1;  /* USED TO PRINT ON LISTING */
        IF BYTE(BUFFER,0)=BYTE('$') THEN /* CONTROL FOR COMPILER */
            DO; OUTPUT=BUFFER; I=1;
               DO WHILE (BYTE(BUFFER,I)=BYTE(' ')) & I<80; I=I+1;
               END;
            J=BYTE(BUFFER,I); CONTROL(J)=¬CONTROL(J);
            IF J=BYTE('A') THEN
               DO; CALL EMITLIT(2); CALL EMITO(SUP);
               END; ELSE
            IF J=BYTE('S') THEN
               DO; IF BLK¬=0 THEN CALL ERROR('$STACK FOLLOWS BEGIN',0);
                  ELSE
                  DO; STACKALL=1; EXSTACK=8192;
                  END;
               END; ELSE
            IF J=BYTE('|') THEN
               IF CONTROL(J) THEN MARGIN_CHOP=TEXT_LIMIT-CP+1; ELSE
               MARGIN_CHOP=0;
            GO TO AGAIN;
            END;
      IF MARGIN_CHOP > 0 THEN
         DO; /* THE MARGIN CONTROL FROM DOLLAR | */
            I = LENGTH(BUFFER) - MARGIN_CHOP;
            REST = SUBSTR(BUFFER, I);
            BUFFER = SUBSTR(BUFFER, 0, I);
         END;
      ELSE REST = '';
      TEXT = BUFFER;
      TEXT_LIMIT = LENGTH(TEXT) - 1;
      IF CONTROL(BYTE('M')) THEN OUTPUT = BUFFER;
      ELSE IF CONTROL(BYTE('L')) THEN
            DO;BUFFER=I_FORMAT(SYLCNT,4)||' |'||BUFFER||'|'||REST;
            BUFFER=I_FORMAT(BLK,2)|| '|'||BUFFER;
            OUTPUT=I_FORMAT(CARD_COUNT,4)||' |'||BUFFER;
            END;
      CP = 0;
   END GET_CARD;
```

```
CHAR:
   PROCEDURE;
      /* USED FOR STRINGS TO AVOID CARD BOUNDARY PROBLEMS */
      CP = CP + 1;
      IF CP <= TEXT_LIMIT THEN RETURN;
      CALL GET_CARD;
   END CHAR;


ENTER: PROCEDURE (C);
    DECLARE C CHARACTER;
    SY=SY+1;
    SYMBOL(SY)=C; TYPE(SY)=0;
    RETURN SY;
    END ENTER;


GETPRT: PROCEDURE(N) FIXED;
    DECLARE N FIXED;
    IF CONTROL(BYTE('V')) THEN OUTPUT='-PRT '||PRTCNT;
    PRTCNT=PRTCNT+1;
    IF PRTCNT>MAXPRT THEN
        DO; CALL ERROR('PRT OVERFLOW',2); PRTCNT=0;
        /* MIGHT CHANGE TO TEST PRTCNT AGAINST MAXDATA-DATACNT */
        END;
     ELSE IF PRTCNT > PRTMAX THEN PRTMAX=PRTCNT;
    DATA(PRTCNT-1)=N; RETURN PRTCNT-1;
    END GETPRT;
SUB_ENTRY: PROCEDURE;
    ARRCNT=0; /* COUNT THE PARAMETERS */
    CALL ENTER(BCD);
    I16=SY;
        BLK=BLK+1;BLOCK(BLK)=SHL(1,31)|SHL(SY,12);
        CALL EMIT0(IM2); CALL EMIT2(0); CALL EMIT0(BRS);
        /* CODE EMITTED TO BRANCH AROUND DECLARATION */
        I1=GETPRT(0); LOCATION(SY)=I1|SHL(SYLCNT,12); SYLEVEL=SY;
    END SUB_ENTRY;
SCAN:
   PROCEDURE;
      DECLARE (S1, S2) FIXED;
      BCD = '';  NUMBER_VALUE = 0;
   SCAN1:
      DO FOREVER;
         IF CP > TEXT_LIMIT THEN CALL GET_CARD;
         ELSE
            DO; /* DISCARD LAST SCANNED VALUE */
               TEXT_LIMIT = TEXT_LIMIT - CP;
```

```
      TEXT = SUBSTR(TEXT, CP);
      CP = 0;
   END;
/*  BRANCH ON NEXT CHARACTER IN TEXT                          */
DO CASE CHARTYPE(BYTE(TEXT));
   /*  CASE 0  */
;
   /*  CASE 1  */
   /*  BLANK  */
   DO;
      CP = 1;
      DO WHILE BYTE(TEXT, CP) = BYTE(' ') & CP <= TEXT_LIMIT;
         CP = CP + 1;
      END;
      CP = CP - 1;
   END;
   /*  CASE 2  */
   DO; /*  LOCATE THE STRING AND PLACE INTO BCD */
   I5=0;
   TOKEN=STRING; BCD='';
       DO FOREVER;
           DO CP=CP+1 TO TEXT_LIMIT;
           S1=BYTE(TEXT,CP); IF S1=BYTE('"') THEN
                DO; CP=CP+1; RETURN;
                END;
           BCD=BCD||SUBSTR(TEXT,CP,1); /* SLOW, BUT FIX LATER */
           END;
       CALL GET_CARD; /* 'CAUSE WE WENT OFF THEN END OF THE CARD */
       END;
   END;
   /*  CASE 3  */
;  /*  NOT USED IN SKELETON (BUT USED IN XCOM)  */
   /*  CASE 4  */
   DO FOREVER;  /* A LETTER:  IDENTIFIERS AND RESERVED WORDS */
      DO CP = CP + 1 TO TEXT_LIMIT;
         IF NOT_LETTER_OR_DIGIT(BYTE(TEXT, CP)) THEN
            DO;  /* END OF IDENTIFIER  */
               IF CP > 0 THEN BCD = BCD || SUBSTR(TEXT, 0, CP);
               S1 = LENGTH(BCD);
               IF S1 > 1 THEN IF S1 <= RESERVED_LIMIT THEN
                  /* CHECK FOR RESERVED WORDS *7
                  DO I = V_INDEX(S1-1) TO V_INDEX(S1) - 1;
                     IF BCD = V(I) THEN
                        DO;
                           TOKEN = I;
                     IF I=44 THEN I5=2;
                     IF I=48 THEN I5=I5+1;
                           RETURN;
```

```
                        END;
                    END;
                /*  RESERVED WORDS EXIT HIGHER: THEREFORE <IDENTIFIER>*/
                IF BCD='COMMENT' THEN
                    DO; S1=0;
                I5=0;
                        DO WHILE S1¬=BYTE(';'); CALL CHAR;
                        S1=BYTE(TEXT,CP);
                        END;
                    BCD=''; NUMBER_VALUE=0; CP=CP+1; GO TO SCAN1;
                    END;
                TOKEN = IDENT;
    IF V(PARSE_STACK(SP))='ARRAY' THEN TOKEN=ADENTIFIER;
            DO I=1 TO I6;
              IF PSTACK(I)=BCD THEN TOKEN=PIDENT;
            END;
            DO I=1 TO I7;
              IF FSTACK(I)=BCD THEN TOKEN=FIDENT;
            END;
          IF I5=3 THEN
              DO;
                FSTACK(I7)=BCD;I7=I7+1;I5=0;
                CALL SUB_ENTRY;TYPE(SY)=FUNC;
              END;
          IF I5=1 THEN
              DO;
                PSTACK(I6)=BCD;I6=I6+1;I5=0;
                CALL SUB_ENTRY;TYPE(SY)=PROC;
              END;
              RETURN;
          END;
    END;
    /*  END OF CARD  */
    BCD = BCD || TEXT;
    CALL GET_CARD;
    CP = -1;
END;
/*  CASE 5  */
DO;       /*  DIGIT:  A NUMBER  */
    TOKEN = NUMBER;
I5=0;
    DO FOREVER;
       DO CP = CP TO TEXT_LIMIT;
          S1 = BYTE(TEXT, CP);
          IF S1 < "F0" THEN RETURN;
          NUMBER_VALUE = 10*NUMBER_VALUE + S1 - "F0";
       END;
       CALL GET_CARD;
```

```
            END;
         END;
         /*  CASE 6  */
            ;
         /*  CASE 7  */
         DO;       /*  SPECIAL CHARACTERS  */
            TOKEN = TX(BYTE(TEXT));
         I5=0;
            IF TOKEN=SPLAT THEN
              DO; CALL CHAR;
              IF BYTE(TEXT,CP)=BYTE('*') THEN
                DO;
                  TOKEN=SPLAT_SPLAT;
                  CP=2;
                  RETURN;
                END;
              CP=CP-1;
              END;
            IF TOKEN = COLON THEN
              DO; CALL CHAR;
              IF BYTE(TEXT,CP)=BYTE('=')&V(PARSE_STACK(SP-1))¬='FOR' THEN
                DO;
                  TOKEN = COLON_EQUALS;
                  CP=2;
                  RETURN;
                END;
              CP=CP-1;
              END;
            CP = 1;
            RETURN;
         END;
         /*  CASE 8  */
       ;    /*  NOT USED IN SKELETON (BUT USED IN XCOM)  */
        END;      /* OF CASE ON CHARTYPE  */
        CP = CP + 1;  /* ADVANCE SCANNER AND RESUME SEARCH FOR TOKEN  */
     END;
  END SCAN;



 /*                          TIME AND DATE                                */

PRINT_TIME:
   PROCEDURE (MESSAGE, T);
      DECLARE MESSAGE CHARACTER, T FIXED;
```

```
      MESSAGE = MESSAGE || T/360000 || ':' || T MOD 360000 / 6000 || ':'
         || T MOD 6000 / 100 || '.';
      T = T MOD 100;  /* DECIMAL FRACTION */
      IF T < 10 THEN MESSAGE = MESSAGE || '0';
      OUTPUT = MESSAGE || T || '.';
   END PRINT_TIME;

PRINT_DATE_AND_TIME:
   PROCEDURE (MESSAGE, D, T);
      DECLARE MESSAGE CHARACTER, (D, T, YEAR, DAY, M) FIXED;
      DECLARE MONTH(11) CHARACTER INITIAL ('JANUARY', 'FEBRUARY', 'MARCH',
         'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST', 'SEPTEMBER', 'OCTOBER',
         'NOVEMBER', 'DECEMBER'),
      DAYS(12) FIXED INITIAL (0, 31, 60, 91, 121, 152, 182, 213, 244, 274,
         305, 335, 366);
      YEAR = D/1000 + 1900;
      DAY = D MOD 1000;
      IF (YEAR & "3") ¬= 0 THEN IF DAY > 59 THEN DAY = DAY + 1; /* ¬ LEAP YEAR*/
      M = 1;
      DO WHILE DAY > DAYS(M);  M = M + 1;  END;
      CALL PRINT_TIME(MESSAGE || MONTH(M-1) || X1 || DAY-DAYS(M-1) ||  ', '
         || YEAR || '.  CLOCK TIME = ', T);
   END PRINT_DATE_AND_TIME;
```



```
 /*                          INITIALIZATION                                  */


INITIALIZATION:
   PROCEDURE;
      EJECT_PAGE;
 OUTPUT='                    ALGOL-W(SUBSET) COMPILATION            ';
      DOUBLE_SPACE;
      CALL PRINT_DATE_AND_TIME ('TODAY IS ',DATE,TIME);
      DOUBLE_SPACE;
      NUMBER = 46;
      ADENTIFIER=49;
      IDENT=50;
      EOFILE = 22;
      STOPIT(2) = TRUE;
      SEMI = 2;
      STRING = 47;
      BEGINV=55;
      ENDV = 23;
```

```
   FIDENT=51;
   PIDENT=52;
   COLON = 4;
   COLON_EQUALS = 20;
   SPLAT=13;
   SPLAT_SPLAT=21;
   IF IDENT = NT THEN RESERVED_LIMIT = LENGTH(V(NT-1));
   ELSE RESERVED_LIMIT = LENGTH(V(NT));
   V(EOFILE) = 'EOF';
   STOPIT(EOFILE) = TRUE;
   CHARTYPE(BYTE(' ')) = 1;
   I6=7;
ARRCNT=0;I19=0;I18=0;
   I7=3;
   I12=0;
   I15=1;I16=1;
   DO I=3 TO 30;
     FSTACK(I)='                                                                    ';
     PSTACK(I)='                                                                    ';
   END;
   DO I=0 TO 30;
     LABEL_NAME(I)='                                                                ';
     LABEL_LOC(I)=0;
FORLIST(I)=0;
   END;
   PSTACK(1)='UNTRACE';
   PSTACK(2)='TRACE';
   PSTACK(3)='DUMP';
   PSTACK(4)='SKIP';
   PSTACK(5)='PAGE';
   PSTACK(6)='WIDTH';
   FSTACK(1)='APPEND';
   FSTACK(2)='STACK';
   DO I = 0 TO 255;
      NOT_LETTER_OR_DIGIT(I) = TRUE;
   END;
   DO I = 0 TO LENGTH(ALPHABET) - 1;
      J = BYTE(ALPHABET, I);
      TX(J) = I;
      NOT_LETTER_OR_DIGIT(J) = FALSE;
      CHARTYPE(J) = 4;
   END;
   DO I = 0 TO 9;
      J = BYTE('0123456789', I);
      NOT_LETTER_OR_DIGIT(J) = FALSE;
      CHARTYPE(J) = 5;
   END;
   DO I = V_INDEX(0) TO V_INDEX(1) - 1;
```

```
        J = BYTE(V(I));
        TX(J) = I;
        CHARTYPE(J) = 7;
    END;
     CHARTYPE(BYTE('"'))=2;  /* CAUSES SCANNER TO TRANSFER TO CASE 2 */
    /* FIRST SET UP GLOBAL VARIABLES CONTROLLING SCAN, THEN CALL IT */
    CP = 0;  TEXT_LIMIT = -1;
    TEXT = '';
    CONTROL(BYTE('L')) = TRUE;
      PRTMAX=0;
      SY,SYLCNT,PRTCNT,DATACNT,SYLEVEL,BLK=0; BLOCK(0)=1;
      PRTLEV(0)=0;
      /* SET ALLOCATIONS IN THE FSA (MAY BE OVERRIDDEN) */
      EXSTACK=200; STACKALL=0;
      /* INITIALIZE OPCODE TABLE */
      OPCODE =
      /* LITERAL CALL SYLLABLE */  'LIT' ||
      /*  08 ARITHMEITIC OPERATORS */  'ADDMINMULDIVEXPNEGRNDTRU'||
      /* 09 BOOLEAN OPERATORS */  'LSSLEQEQLNEQGEQGTRNOTANDBOR' ||
      /* 4 LOAD/STORE OPERATORS */  'IMIIM2LODSTO' ||
      /* 07 STACK CONTROL */  'PRORTNXITDELDUPXCHINX'||
      /* 07 PROGRAM CONTROL */ 'BFNBBNBFCBBCBRSBSCNOP'||
      /* 14 MONITOR CALLS */ 'GETRETRDVWRVWRSDMPTABSUBBIFROWSUPSAVUNSSTD';
/*INSERT "REM" IN THE PROPER PLACE WHEN THE
  ALGOLE MACHINE IS CHANGED TO ACCOMODATE IT */
  /*   MOVE THE STORE DESTRUCTIVE (LAST OP) TO THE PROPER PLACE LATER */
      /* SET-UP THE OP CODES */
      I2=ADDR(ADD); I3=1;
          DO WHILE I3 <=OPERATORS;
          COREBYTE(I2)=I3; I3=I3+1; I2=I2+1;
          END;
     CALL ENTER('UNTRACE'); TYPE(SY)=IPROC; LOCATION(SY)=0;
     CALL ENTER('TRACE'); TYPE(SY)=IPROC; LOCATION(SY)=1;
     CALL ENTER('DUMP'); TYPE(SY)=IPROC; LOCATION(SY)=2;
     CALL ENTER('SKIP'); TYPE(SY)=IPROC; LOCATION(SY)=SHL(1,27)|3;
     CALL ENTER('PAGE'); TYPE(SY)=IPROC; LOCATION(SY)=4;
     CALL ENTER('APPEND'); TYPE(SY)=IFUNC; LOCATION(SY)=5;
      CALL ENTER('WIDTH'); TYPE(SY)=IPROC; LOCATION(SY)=SHL(1,27)|6;
      CALL ENTER('STACK'); TYPE(SY)=IFUNC; LOCATION(SY)=7;
     /* IF THE INTRINSIC REQUIRES PARAMETERS THEN SHL(#PARAMS,27) INTO
     LOCATION(SY) */
     OUTPUT='CARD |BL| SYL |';
    CALL SCAN;
    /* INITIALIZE THE PARSE STACK */
    SP = 1;  PARSE_STACK(SP) = EOFILE;
END INITIALIZATION;
```

```
DUMPIT:
   PROCEDURE;     /* DUMP OUT THE STATISTICS COLLECTED DURING THIS RUN  */
      DOUBLE_SPACE;
      /*  PUT OUT THE ENTRY COUNT FOR IMPORTANT PROCEDURES */

      OUTPUT = 'STACKING DECISIONS= ' || CALLCOUNT(1);
      OUTPUT = 'SCAN               = ' || CALLCOUNT(3);
      OUTPUT = 'FREE STRING AREA   = ' || FREELIMIT - FREEBASE;
   END DUMPIT;


STACK_DUMP:
   PROCEDURE;
      DECLARE LINE CHARACTER;
      LINE = 'PARTIAL PARSE TO THIS POINT IS: ';
      DO I = 2 TO SP;
         IF LENGTH(LINE) > 105 THEN
            DO;
               OUTPUT = LINE;
               LINE = X4;
            END;
         LINE = LINE || X1 || V(PARSE_STACK(I));
      END;
      OUTPUT = LINE;
   END STACK_DUMP;

LOOKUP: PROCEDURE(C);
    DECLARE C CHARACTER, I FIXED;
        DO I=0 TO SY-1; IF SYMBOL(SY-I)=C THEN RETURN SY-I;
        END;
    RETURN 0;
    END LOOKUP;

TRACER: PROCEDURE(P);
    DECLARE P FIXED, L CHARACTER;
    L=SUBSTR(X70,0,15);
        DO I1=LEFT TO RIGHT; L=L||V(PARSE_STACK(I1))||' ';
        END;
    OUTPUT=L;
    END TRACER;

GETSTORAGE: PROCEDURE(N) FIXED;
    DECLARE (N,I) FIXED;
    I=DATACNT;
```

```
     IF CONTROL(BYTE('F')) THEN OUTPUT='-FIX '||I;
     DATACNT=DATACNT+N;
     IF DATACNT > MAXDATA-PRTCNT THEN
          DO; CALL ERROR ('DATA TABLE OVERFLOW',2); I,DATACNT=0;
          END;
          DO N=MAXDATA-DATACNT TO MAXDATA+I-1;
          DATA(N)=0;
          END;
     RETURN I;
     END GETSTORAGE;

STORESTRING: PROCEDURE(S) FIXED;
     DECLARE S CHARACTER, (I,J,K,L,M,T) FIXED;
     J=LENGTH(S);
     I=GETSTORAGE( SHR(J+4,2) );
     IF SEVERE_ERRORS=0 THEN /* LOAD STRING INTO DATA ARRAY */
          DO; T=SHL(J,24); L=1; M=MAXDATA-I;
              DO K=0 TO J-1;
              T=T|SHL(BYTE(S,K),SHL(3-L,3));
              L=(L+1) & "3";
              IF L=0 THEN
                   DO; DATA(M)=T; T=0;  M=M-1;
                   END;
              END;
          IF L¬=0 THEN DATA(M)=T;
          END;
     RETURN I;
     END STORESTRING;

ARRAYIDENT: PROCEDURE;
     DECLARE (N,P) BIT(16);
          CALL ENTER(VAR(LEFT+2));TYPE(SY)=ARRAY;
          P=GETPRT(0);LOCATION(SY)=P;
          ARRCNT=ARRCNT+1;
     END ARRAYIDENT;

SUB_PARAM: PROCEDURE;
     I1=LOOKUP(VAR(LEFT+1));
     CALL ENTER(VAR(LEFT+1));
     TYPE(SY)=SIMPLE; LOCATION(SY)=GETPRT(0); ARRCNT=ARRCNT+1;
     END SUB_PARAM;

SUB_INIT: PROCEDURE;
     I1=BLOCK(BLK);
          DO; I2=SHR(I1,12)&"3FF"; I3=LOCATION(I2); I4=ARRCNT;
          /* I3&"FFF" IS PRT CELL OF PROC/FUNC NAME, I4 IS PARAM COUNT */
          CALL EMITLIT(I4); CALL EMITO(IM2); I17=SYLCNT;
          CALL EMIT2(0); CALL EMITLIT(I3&"FFF"); CALL EMITO(SAV);
```

```
        END;
    END SUB_INIT;


  /*                    THE SYNTHESIS ALGORITHM FOR XPL                    */

SYNTHESIZE: PROCEDURE(PRODUCTION_NUMBER,REDUCTION) FIXED;
    DECLARE (PRODUCTION_NUMBER,REDUCTION) FIXED;
    IF CONTROL(BYTE('P')) THEN CALL TRACER(PRODUCTION_NUMBER);
  DO CASE PRODUCTION_NUMBER;
    ;
 /*  <PROGRAM> ::= <BLOCK> .     */
 DO; COMPILING=FALSE;
        /* CLEAR THE CODE ACCUMULATION WORD (CODEW) */
        I1=4-(SYLCNT & "3");
            DO I2=0 TO I1;
            CALL EMITO(XIT);
            END;
      IF MP ¬= 2 THEN  /* WE DIDN'T GET HERE LEGITIMATELY  */
         DO;
            CALL ERROR ('EOF AT INVALID POINT', 1);
            CALL STACK_DUMP;
         END;
   END;
 /*   <BLOCK> ::= <BLOCK HEAD> <BLOCK END>     */
       ENDS:
        DO; I4=BLOCK(BLK);
            DO WHILE SHR(I4,30)¬=0;
            BLK=BLK-1; I4=BLOCK(BLK);
            END;
        IF PRTCNT>PRTLEV(BLK) THEN PRTLEV(BLK)=PRTCNT;
        I3=BLK-1; I2=BLOCK(I3);
        IF SHR(I2,30)>0 THEN /* CASE STMT OR PROC DEFINITION */
            DO;
            IF SHR(I2,30)=1 THEN /* CASE STMT */
                DO; I3=I3-1; I2=BLOCK(I3);
                END;
            IF SHR(I2,31)=1 THEN /* PROCEDURE DEFINITION */
                DO; I3=I3-1; I2=BLOCK(I3); PRTCNT=PRTLEV(BLK);
                END; ELSE PRTCNT=I4&"FFF";
            END; ELSE
        PRTCNT=I4&"FFF";
        IF PRTLEV(BLK) > PRTLEV(I3) THEN PRTLEV(I3)=PRTLEV(BLK);
        /* RESTORE SYMBOL TABLE LEVEL */
        I1=0;
        BLK=BLK-1; SYLEVEL=SHR(I4,12) & "3FF";
```

```
            DO WHILE SY>SYLEVEL;
            I2=TYPE(SY);
            IF I2=ARRAY THEN
                DO; IF STACKALL THEN I1=SY; ELSE
                    DO; CALL EMITLIT(LOCATION(SY)&"FFF"); CALL EMITO(LOD);
                    CALL EMITO(RET);
                    END;
                END;
            SY=SY-1;
            END;
        IF I1>0 THEN
            DO; CALL EMITLIT(LOCATION(I1) & "FFF"); CALL EMITO(LOD);
            CALL EMITO(RET);
            END;
        I4=BLOCK(BLK); SYLEVEL=SHR(I4,12)&"3FF";
        END;
/*  <BLOCK HEAD> ::= <PROLOGUE>    */
;
/*  <BLOCK HEAD> ::= <BLOCK HEAD> <LABEL IDENTIFIER> <STATEMENT> ;    */
   GO TO CHECK_CASE;
/*  <BLOCK HEAD> ::= <BLOCK HEAD> <LABEL IDENTIFIER> ;    */
;
/*  <BLOCK HEAD> ::= <BLOCK HEAD> <STATEMENT> ;    */
      CHECK_CASE:
      DO; I1=BLOCK(BLK-1);
      IF SHR(I1,30)=1 THEN
          DO; /* IN A CASE STMT */ CALL EMITO(IM2);
          CALL EMIT2(SHR(I1,15)&"7FFF"); CALL EMITO(BRS);
          I2=(I1&"7FFF")+1; BLOCK(BLK-1)=SHL(1,30)|SHL(SYLCNT,15)|I2;
          END;
      END;
/*  <BLOCK HEAD> ::= <BLOCK HEAD> ;    */
;
/*  <PROLOGUE> ::= <BEGIN>    */
;
/*  <PROLOGUE> ::= <BLOCK HEAD> <DECLARATION SET> ;    */
;
/*  <BEGIN> ::= BEGIN    */
      DO;BLK=BLK+1;SYLEVEL=SY;BLOCK(BLK)=SHL(SY,12)|PRTCNT;
        PRTLEV(BLK)=PRTCNT;
      END;
/*  <DECLARATION SET> ::= <DECLARATION>    */
;
/*  <DECLARATION> ::= <SIMPLE VARIABLE DECLARATION>    */
;
/*  <DECLARATION> ::= <PROCEDURE DECLARATION>    */
      DO; I1=BLOCK(BLK); I2=SHR(I1,12)&"3FF"; I1=I1&"FFF";
      /* I1 IS THE PRT COUNT BEFORE THE LOCALS WERE ENCOUNTERED, I2 IS
```

```
        THE LOCATION OF THE PROC/FUNC IN THE SYMBOL TABLE */
        /* RESTORE THE TOP OF THE SYMBOL TABLE */  SY=I2;
        I1=PRTCNT-I1; /* I1 IS NOW THE NUMBER OF LOCALS */
        I1=I1+I18;I18=0;
        I2=LOCATION(I2); I3=SHR(I2,27); /* I3 IS NUMBER OF PARAMETERS */
        CALL EMITLIT(I1+I3); CALL EMITLIT(I2&"FFF"); CALL EMITO(UNS);
        CALL EMITO(RTN); /* NOW BACK STUFF #LOCALS AND BRANCH AROUND DECLAR */
        I3=SHR(SYLCNT,2); CODE(I3)=CODEW;
        CALL STUFFBACK(I17,I1);I1=(SHR(I2,12)&"7FFF")-3;
        CALL STUFFBACK(I1,SYLCNT); CODEW=CODE(I3); I1=GETPRT(0);
        /* INITIALIZE THE BASE BLOCK POINTER */
        CALL EMITLIT(I1); CALL EMITLIT(0); CALL EMITO(STD);
        BLK=BLK-1; I1=BLOCK(BLK); SYLEVEL=SHR(I1,12)&"3FF";
        END;
/*  <DECLARATION> ::= <ARRAY DECLARATION>     */
;
/* <ARRAY DECLARATION> ::= INTEGER ARRAY <ADENTIFIER>
     <BOUND PAIR LIST>    */
      DO; FIXV(LEFT+1)=0;ARRCNT=0;
        CALL ARRAYIDENT;
        I1=ARRCNT; I2=FIXV(LEFT);
        /* I1 IS NUMBER OF ARRAYS TO BE ALLOCATED, FIXV(RIGHT) IS NUMBER
        OF DIMENSIONS */
        ARRCNT=FIXV(RIGHT); CALL EMITLIT(ARRCNT);
        CALL EMITLIT(I1); CALL EMITO(ROW); /* ARRAY NOW SET-UP */
        /* NOW FILL PRT CELLS */
            DO I3=1 TO I1; I2=SY-I3+1;
            I1=LOCATION(I2); LOCATION(I2)=SHL(ARRCNT,12) | I1;
            CALL EMITLIT(I1); CALL EMITO(XCH); CALL EMITO(STD);
            END;
        END;
/*  <BOUND PAIR LIST> ::= <BOUND PAIR HEAD> <BOUND PAIR> )     */
   FIXV(LEFT)=FIXV(LEFT)+1; /* COUNT SUBSCRIPTS */
/*  <BOUND PAIR> ::= <EXPRESSION> : : <EXPRESSION>     */
;
/*  <BOUND PAIR HEAD> ::= <BOUND PAIR HEAD> <BOUND PAIR> ,     */
      FIXV(LEFT)=FIXV(LEFT)+1;
/*  <BOUND PAIR HEAD> ::= (     */
      FIXV(LEFT)=0; /* INITIALIZE SUBSCRIPT COUNT */
/*  <SIMPLE VARIABLE DECLARATION> ::= <TYPE HEAD> <IDENTIFIER>     */
      DO; I2=RIGHT; GO TO ADDLOCAL;
      END;
/*  <TYPE HEAD> ::= <TYPE HEAD> <IDENTIFIER> ,     */
      DO; I2=LEFT+1; ADDLOCAL:
      I1=LOOKUP(VAR(I2));
          DO; CALL ENTER(VAR(I2)); TYPE(SY)=SIMPLE; LOCATION(SY)=GETPRT(0);
          END;
      END;
```

```
/*  <TYPE HEAD> ::= <INTEGER>    */
;
/*  <INTEGER> ::= INTEGER    */
;
/*  <PROCEDURE DECLARATION> ::= <PROPER PROC DECL>    */
;
/*  <PROCEDURE DECLARATION> ::= <FUNCTION PROC DECL>    */
;
/*  <PROPER PROC DECL> ::= <PROCEDURE> <PROC HEAD> ; <STATEMENT>    */
;
/*  <PROCEDURE> ::= PROCEDURE    */
  I19=0;
/*  <FUNCTION PROC DECL> ::= INTEGER PROCEDURE <PROC HEAD> ;
     <FUNCTION PROC BODY>    */
;
/*  <FUNCTION PROC BODY> ::= <EXPRESSION>    */
 DO;
     I16=LOCATION(I16);
     CALL EMITLIT(I16 & "FFF");
     CALL EMITO(XCH);CALL EMITO(STD);
 END;
/*  <FUNCTION PROC BODY> ::= <BLOCK HEAD> <EXPRESSION> END    */
 DO;
     I16=LOCATION(I16);
     CALL EMITLIT(I16 & "FFF");
     CALL EMITO(XCH);CALL EMITO(STD);
   GO TO ENDS;
 END;
/*  <PROC HEAD> ::= <IDENTIFIER>    */
   DO;
       I1=BLOCK(BLK); BLOCK(BLK)=I1|PRTCNT;
       CALL SUB_INIT;
   END;
/*  <PROC HEAD> ::= <IDENTIFIER> ( <FORMAL PARAMETER LIST> )    */
       CALL SUB_INIT;
/*  <FORMAL PARAMETER LIST> ::= <FORMAL PARAMETER SEGMENT>    */
;
/*  <FORMAL PARAMETER LIST> ::= <FORMAL PARAMETER LIST> ;
     <FORMAL PARAMETER SEGMENT>    */
;
/*  <FORMAL PARAMETER SEGMENT> ::= <FORMAL TYPE> <IDENTIFIER>    */
 DO;
       CALL SUB_PARAM;I1=BLOCK(BLK);BLOCK(BLK)=I1|PRTCNT;
       I1=LOCATION(SYLEVEL); LOCATION(SYLEVEL)=I1|SHL(ARRCNT,27);
 END;
/*  <FORMAL TYPE> ::= <FORMAL TYPE> <IDENTIFIER> ,    */
       CALL SUB_PARAM;
/*  <FORMAL TYPE> ::= <INTEGER> VALUE    */
```

```
;
/*  <BLOCK END> ::= <STATEMENT> END    */
   GO TO CHECK_CASE;
/*  <BLOCK END> ::= <LABEL IDENTIFIER> <STATEMENT> END    */
      GO TO CHECK_CASE;
/*  <BLOCK END> ::= <LABEL IDENTIFIER> END    */
;
/*  <BLOCK END> ::= END    */
;
/*  <LABEL IDENTIFIER> ::= <IDENTIFIER> :    */
    DO; CALL ENTER(VAR(LEFT));TYPE(SY)=LABELV;LOCATION(SY)=GETPRT(0);
      LABEL_NAME(I12)=VAR(LEFT);
      LABEL_LOC(I12)=SYLCNT;I12=I12+1;
    END;
/*  <STATEMENT> ::= <SIMPLE STATEMENT>    */
;
/*  <STATEMENT> ::= <ITERATIVE STATEMENT>    */
 IF I21=1 THEN
 DO;I21=0;
 I1=SHR(SYLCNT,2);CODE(I1)=CODEW;
 CALL STUFFBACK(I22,I9);
 END;
/*  <STATEMENT> ::= <IF STATEMENT>    */
;
/*  <STATEMENT> ::= <CASE STATEMENT>    */
;
/*  <CASE STATEMENT> ::= <CASE CLAUSE> <BLOCK>    */
      DO;  I1=BLOCK(BLK);
      IF SHR(I1,30)=1 THEN /* MAY HAVE BEEN A SYNTAX ERROR OTHERWISE */
          DO;  I2=SHR(SYLCNT,2); CODE(I2)=CODEW;
          /* GET NUMBER OF STATEMENTS IN THE CASE */ I3=I1&"7FFF";
CALL STUFFBACK(I11,I3);
          I4=(SHR(I1,15)&"7FFF")-3; /* FIRST ADDRESS TO STUFF */
          /* MULT NUMBER OF STATEMENTS BY FOUR AND ADD TO SYLCNT */
         I1=SYLCNT+SHL(I3,2); /* RUN THROUGH BACK CHAIN AND STUFF */
          CALL STUFFBACK(FIXV(LEFT),I1 -4); /* BRNCH TO CASE SELECTOR*/
              DO WHILE I3>0; I3=I3-1; I4=STUFFBACK(I4,I1);
              IF SHR(SYLCNT,2)=I2 THEN CODEW=CODE(I2);
              CALL EMITO(IM2); CALL EMIT2(I4); CALL EMITO(BRS); I4=I4-3;
              /* I4 POINTS TO NEXT ADDRESS TO BACKSTUFF */
              END;
          BLK=BLK-1;
          END;
      END;
/*  <CASE CLAUSE> ::= CASE <EXPRESSION> OF    */
   DO;
CALL EMITO(DUP);
CALL EMITLIT(1);
```

```
 CALL EMITO(GEQ);
 CALL EMITO(IM2);
  I8=SYLCNT;
 CALL EMIT2(0);
 CALL EMITO(BSC);
 CALL EMITO(DUP);
 CALL EMITO(IM2);
  I11=SYLCNT;
 CALL EMIT2(0);
 CALL EMITO(GTR);
 CALL EMITO(IM2);
  I9=SYLCNT;
 CALL EMIT2(0);
 CALL EMITO(BSC);
  CALL STUFFBACK(I8,SYLCNT);
I10=STORESTRING('*ERROR* INVALID CASE INDEX AT SYLLABLE ');
 CALL EMITLIT(I10);
 CALL EMITO(INX);
 CALL EMITO(WRS);
 CALL EMITLIT(SYLCNT);
 CALL EMITO(WRV);
 CALL EMITO(DMP);
 CALL EMITO(XIT);
  CALL STUFFBACK(I9,SYLCNT);
 CALL EMITO(NOP);
 CALL EMITLIT(4);CALL EMITO(MUL);CALL EMITO(IM2);
        FIXV(LEFT)=SYLCNT; CALL EMIT2(0); /* BACK STUFF LATER */
       CALL EMITO(XCH);CALL EMITO(MIN);CALL EMITLIT(4);CALL EMITO(ADD);
       CALL EMITO(BRS); /* BRANCH TO CASE
        SELECTION CODE (LOCATED AFTER THE CASES) */
        IF BLK=MAXLEVEL THEN CALL ERROR('MAXIMUM BLOCK LEVEL EXCEEDED',2); ELSE
            DO; BLK=BLK+1; BLOCK(BLK)=SHL(1,30)|SHL(SYLCNT,15);
            END;
        END;
 /*  <IF STATEMENT> ::= <IF CLAUSE> <STATEMENT>     */
        DO; I1=SHR(SYLCNT,2); CODE(I1)=CODEW;
        CALL STUFFBACK(FIXV(LEFT),SYLCNT); CODEW=CODE(I1);
        END;
 /*  <IF STATEMENT> ::= <IF CLAUSE> <SIMPLE STATEMENT> ELSE <STATEMENT>     */
   DO;
        DO; CALL EMITO(IM2);FIXV(LEFT)=SYLCNT;
        CALL EMIT2(0);
        CALL EMITO(BRS); I2=SHR(SYLCNT,2); CODE(I2)=CODEW;
        CALL EMITO(BRS);CALL EMITO(DEL);
        END;
        DO; I1=SHR(SYLCNT,2); CODE(I1)=CODEW;
        CALL STUFFBACK(FIXV(LEFT),SYLCNT); CODEW=CODE(I1);
        END;
```

```
END;
/*  <IF CLAUSE> ::= IF <LOGICAL EXPRESSION> THEN    */
      DO;CALL EMITO(IM2);I13=SYLCNT;CALL EMIT2(0);CALL EMITO(BSC);
      FIXV(LEFT)=SYLCNT-3;
      END;
/*  <LOGICAL EXPRESSION> ::= <LOGICAL ELEMENT>    */
;
/*  <LOGICAL EXPRESSION> ::= <RELATION>    */
;
/*  <RELATION> ::= <SIMPLE EXPRESSION> <EQUALITY OPERATOR> <SIMPLE EXPRESSION>
      */
   CALL EMITO(FIXV(LEFT+1));
/*  <RELATION> ::= <LOGICAL ELEMENT> <EQUALITY OPERATOR> <LOGICAL ELEMENT>
      */
   CALL EMITO(FIXV(LEFT+1));
/*  <RELATION> ::= <SIMPLE EXPRESSION> <RELATIONAL OPERATOR>
     <SIMPLE EXPRESSION>    */
   CALL EMITO(FIXV(LEFT+1));
/*  <EQUALITY OPERATOR> ::= =    */
   FIXV(LEFT)=EQL;
/*  <EQUALITY OPERATOR> ::= ¬ =    */
   FIXV(LEFT)=NEQ;
/*  <RELATIONAL OPERATOR> ::= <    */
   FIXV(LEFT)=LSS;
/*  <RELATIONAL OPERATOR> ::= < =    */
   FIXV(LEFT)=LEQ;
/*  <RELATIONAL OPERATOR> ::= >    */
   FIXV(LEFT)=GTR;
/*  <RELATIONAL OPERATOR> ::= > =    */
   FIXV(LEFT)=GEQ;
/*  <LOGICAL ELEMENT> ::= <LOGICAL TERM>    */
;
/*  <LOGICAL ELEMENT> ::= <LOGICAL ELEMENT> OR <LOGICAL TERM>    */
   CALL EMITO(BOR);
/*  <LOGICAL TERM> ::= <LOGICAL FACTOR>    */
;
/*  <LOGICAL TERM> ::= <LOGICAL TERM> AND <LOGICAL FACTOR>    */
        CALL EMITO(AND);
/*  <LOGICAL FACTOR> ::= <LOGICAL PRIMARY>    */
;
/*  <LOGICAL FACTOR> ::= ¬ <LOGICAL PRIMARY>    */
        CALL EMITO(NOT);
/*  <LOGICAL PRIMARY> ::= <LOGICAL VALUE>    */
;
/*  <LOGICAL PRIMARY> ::= ( <LOGICAL EXPRESSION> )    */
;
/*  <LOGICAL VALUE> ::= TRUE    */
;
```

```
/*  <LOGICAL VALUE> ::= FALSE    */
;
/*  <ITERATIVE STATEMENT> ::= <FOR CLAUSE> <STATEMENT>    */
DO;
   I1=1;
 IF FORLIST(1) ¬ = 0 THEN
 DO I=1 TO I15-1;
   CALL EMITLIT(LOCATION(LOOKUP(S1)));
   CALL EMITO(LOD);
   CALL EMITLIT(I1);
   CALL EMITO(NEQ);
   CALL EMITLIT(FORLIST(I)+3);
   CALL EMITO(BSC);
   I1=I1+1;
 END;ELSE
 DO;
 CALL EMITO(IM2);
 I21=1;I22=SYLCNT;
 CALL EMIT2(0);
 CALL EMITO(BRS);CALL EMITO(DEL);
 CALL EMITO(BRS);
 END;
   CALL STUFFBACK(FIXV(LEFT),SYLCNT);
   PRTCNT=PRTCNT-3;
   SY=SY-3;
 DO I=1 TO  I15;
   FORLIST(I)=0;
 END;
   I15=1;
END;
/*   <ITERATIVE STATEMENT> ::= <WHILE CLAUSE> <STATEMENT>    */
       DO; CALL EMITO(IM2);FIXV(RIGHT)=SYLCNT; CALL EMIT2(0); CALL EMITO(BRS);
           I1=SHR(SYLCNT,2); CODE(I1)=CODEW; I2=STUFFBACK(FIXV(LEFT),SYLCNT);
       /* BRANCH AROUND THE STMT IS NOW STUFFED */
       CALL STUFFBACK(FIXV(RIGHT),I2); /* BRANCH BACK TO TEST IS STUFFED */
       CODEW=CODE(I1);
       END;
/*  <WHILE CLAUSE> ::= WHILE <LOGICAL EXPRESSION> DO    */
       DO; CALL EMITO(IM2); I1=SYLCNT; CALL EMIT2(I14);
       FIXV(LEFT)=I1; CALL EMITO(BSC);
       END;
/*  <FOR CLAUSE> ::= <FOR HEAD> <FOR END>    */
   ;
/*  <FOR HEAD> ::= FOR <IDENTIFIER> : =    */
DO;
  S=VAR(LEFT+1);
   S1='XXXXX';S2='YYYYY';
   CALL ENTER(S);LOCATION(SY)=GETPRT(0);
```

```
    TYPE(SY)=SIMPLE;
    CALL ENTER(S1);LOCATION(SY)=GETPRT(0);
    TYPE(SY)=SIMPLE;
    CALL ENTER(S2);LOCATION(SY)=GETPRT(0);
    TYPE(SY)=SIMPLE;
        I18=3;
  END;
/*  <FOR END> ::= <FOR LIST> DO    */
  DO;
    CALL EMITO(IM2);
    FIXV(LEFT)=SYLCNT;
    CALL EMIT2(0);
    CALL EMITO(BRS);
    DO I=1 TO I15-1;
      CALL STUFFBACK(FORLIST(I),SYLCNT);
    END;
  END;
/*  <FOR END> ::= <EXPRESSION> <INCREMENT> UNTIL <EXPRESSION> DO    */
DO;
  CALL EMITLIT(LOCATION(LOOKUP(S2)));
  CALL EMITO(XCH);
  CALL EMITO(STD);
  CALL EMITLIT(LOCATION(LOOKUP(S1)));
  CALL EMITO(XCH);
  CALL EMITO(STD);
  CALL EMITLIT(LOCATION(LOOKUP(S)));
  CALL EMITO(XCH);
  CALL EMITO(STO);
  CALL EMITO(IM2);
  FIXV(LEFT+1)=SYLCNT;
  CALL EMIT2(0);
  CALL EMITO(BRS);
  I9=SYLCNT;
  CALL EMITLIT(LOCATION(LOOKUP(S1)));
  CALL EMITO(LOD);
  CALL EMITLIT(LOCATION(LOOKUP(S)));
  CALL EMITO(LOD);
  CALL EMITO(ADD);
  CALL EMITLIT(LOCATION(LOOKUP(S)));
  CALL EMITO(XCH);
  CALL EMITO(STO);
  CALL STUFFBACK(FIXV(LEFT+1),SYLCNT);
  CALL EMITLIT(LOCATION(LOOKUP(S2)));
  CALL EMITO(LOD);
  CALL EMITO(LEQ);
  CALL EMITO(IM2);
    FIXV(LEFT)=SYLCNT;
  CALL EMIT2(0);
```

```
 CALL EMITO(BSC);
END;
/*  <FOR END> ::= <EXPRESSION> UNTIL <EXPRESSION> DO    */
DO;
 CALL EMITLIT(LOCATION(LOOKUP(S2)));
 CALL EMITO(XCH);
 CALL EMITO(STD);
 CALL EMITLIT(LOCATION(LOOKUP(S)));
 CALL EMITO(XCH);
 CALL EMITO(STO);
 CALL EMITO(IM2);
 FIXV(LEFT+1)=SYLCNT;
 CALL EMIT2(0);
 CALL EMITO(BRS);
 I9=SYLCNT;
 CALL EMIT1(1);
 CALL EMITLIT(LOCATION(LOOKUP(S)));
 CALL EMITO(LOD);
 CALL EMITO(ADD);
 CALL EMITLIT(LOCATION(LOOKUP(S)));
 CALL EMITO(XCH);
 CALL EMITO(STO);
 CALL STUFFBACK(FIXV(LEFT+1),SYLCNT);
 CALL EMITLIT(LOCATION(LOOKUP(S2)));
 CALL EMITO(LOD);
 CALL EMITO(LEQ);
 CALL EMITO(IM2);
   FIXV(LEFT)=SYLCNT;
 CALL EMIT2(0);
 CALL EMITO(BSC);
END;
/*  <INCREMENT> ::= STEP <EXPRESSION>    */
;
/*  <FOR LIST> ::= <EXPRESSION>    */
  GO TO LISTFOR
;
/*  <FOR LIST> ::= <FOR LIST> , <EXPRESSION>    */
LISTFOR: DO;
 CALL EMITLIT(LOCATION(LOOKUP(S)));
 CALL EMITO(XCH);
 CALL EMITO(STO);
 CALL EMITLIT(LOCATION(LOOKUP(S1)));
 CALL EMITO(LOD);
 CALL EMIT1(1);
 CALL EMITO(ADD);
 CALL EMITLIT(LOCATION(LOOKUP(S1)));
 CALL EMITO(XCH);
 CALL EMITO(STO);
```

```
 CALL EMITO(IM2);
 FORLIST(I15)=SYLCNT;
 I15=I15+1;
 CALL EMIT2(0);
 CALL EMITO(BRS);
END;
/*  <SIMPLE STATEMENT> ::= <ASSIGNMENT STATEMENT>     */
          DO; /* CHANGE THE STO TO STD */
          CALL BACKUP; CALL EMITO(STD);
          END;
/*  <SIMPLE STATEMENT> ::= <PROCEDURE STATEMENT>     */
    CALL EMITO(DEL);
/*  <SIMPLE STATEMENT> ::= <GOTO STATEMENT>     */
;
/*  <SIMPLE STATEMENT> ::= <BLOCK>     */
;
/*  <SIMPLE STATEMENT> ::= <READ STATEMENT>     */
;
/*  <SIMPLE STATEMENT> ::= <WRITE STATEMENT>     */
;
/*  <GOTO STATEMENT> ::= GOTO <IDENTIFIER>     */
    LABEL_IDENTIFIER:
    DO I=0 TO I12;
    IF LOOKUP(VAR(RIGHT)) ¬= 0 THEN
      IF VAR(RIGHT)=LABEL_NAME(I) THEN
      DO;CALL EMITLIT(LABEL_LOC(I));CALL EMITO(BRS);
      END;
    END;
/*  <GOTO STATEMENT> ::= GO TO <IDENTIFIER>     */
    GO TO LABEL_IDENTIFIER;
/*  <PROCEDURE STATEMENT> ::= <CALL HEADING> <CALL BODY> )     */
      DO; I1=FIXV(LEFT)+1; I2=SHR(I1,16); I1=I1&"FFFF";
      /* I1 IS NUMBER OF PARAMS, I2 IS SYMBOL TABLE LOCATION */
          I3,FIXV(LEFT)=TYPE(I2);I4=I2;I2=LOCATION(I2);
      IF I3>=IFUNC&I3<=IPROC THEN /*INTRINSIC*/
        DO;CALL EMITLIT(I2&"7FFF");CALL EMITO(BIF);
        END; ELSE
        DO; /*USER DEFINED */
          CALL EMITLIT(SHR(I2,12)&"7FFF");CALL EMITO(PRO);
        END;
      END;
/*  <PROCEDURE STATEMENT> ::= <PROCEDURE IDENTIFIER>     */
 DO; I2=TYPE(LOOKUP(VAR(LEFT)));
   I3=LOCATION(LOOKUP(VAR(LEFT)));
   IF I2>=IFUNC&I2<=IPROC THEN /*INTRINSIC*/
     DO;CALL EMITLIT(I3&"7FFF");CALL EMITO(BIF);
     END;ELSE /*USER DEFINED*/
DO;CALL EMITLIT(SHR(I3,12)&"7FFF");CALL EMITO(PRO);
```

```
      END;
  END;
/*  <CALL HEADING> ::= <PROCEDURE IDENTIFIER> (     */
    DO;I1=LOOKUP(VAR(LEFT));I2=TYPE(I1);
      FIXV(LEFT)=SHL(I1,16);
    END;
/*   <CALL HEADING> ::= <CALL HEADING> <CALL BODY> ,     */
      FIXV(LEFT)=FIXV(LEFT)+1;
/*   <CALL BODY> ::= <STATEMENT>     */
;
/*   <CALL BODY> ::= <EXPRESSION>     */
;
/*   <ASSIGNMENT STATEMENT> ::= <LEFT PART> <EXPRESSION>     */
          CALL EMITO(STO); /* MIGHT BE CHANGED TO DESTRUCTIVE STORE */
/*   <ASSIGNMENT STATEMENT> ::= <LEFT PART> <ASSIGNMENT STATEMENT>     */
;
/*   <LEFT PART> ::= <VARIABLE> :=     */
          DO; I1=FIXV(LEFT);
          IF I1¬=0 THEN /* VARIABLE OR FUNCTION NAME */
              DO;
              IF I1 < 0 THEN /* FUNCTION NAME */
                  DO; I1=-I1; FIXV(LEFT)=0; /* SO IT CANNOT BE USED
                  IN A FOR STATEMENT */
                      DO; I2=LOCATION(I1); CALL EMITLIT( I2&"FFF");
                      CALL EMITO(XCH);
                      END;
                  END;
              END;
          END;
/*   <VARIABLE> ::= <ARRAY DESIGNATOR>     */
          FIXV(LEFT)=0; /* MARK AS NOT VARIABLE OR FUNC */
/*   <VARIABLE> ::= <IDENTIFIER>     */
          DO; I1=LOOKUP(VAR(LEFT));
             I2=TYPE(I1);
              IF I2=SIMPLE THEN CALL EMITLIT(LOCATION(I1)); ELSE
              I1=-I1; /* PROCESS FUNCS AND PROCS LATER */
          FIXV(LEFT)=I1;
          END;
/*   <ARRAY DESIGNATOR> ::= <IDENTIFIER> <EXPRESSION LIST> <EXPRESSION> )     */
    DO;I1=LOOKUP(VAR(LEFT));FIXV(LEFT+1)=SHL(I1,16);
        FIXV(LEFT+1)=FIXV(LEFT+1)+1;
        /* FIXV CONTAINS THE NUMBER OF SUBSCRIPTS ENCOUNTERED */
      I1=FIXV(LEFT+1);I2=I1&"FFFF";I1=SHR(I1,16);I2=I2+1;
      /* I1 IS THE SYMBOL TABLE LOCATION, I2 IS THE SUBSCRIPT COUNT */
      I3=LOCATION(I1); CALL EMITLIT(I3&"FFF"); CALL EMITO(LOD);
      /* PRT LOCATION IS NOW AT THE TOP OF THE STACK */
      CALL EMITO(SUB);
      /* NOW HAVE ABS ADDR AT TOP OF STACK */
```

```
      END;
/*  <EXPRESSION LIST> ::= (    */
;
/*  <EXPRESSION LIST> ::= <EXPRESSION LIST> <EXPRESSION> ,    */
   FIXV(LEFT)=FIXV(LEFT)+1; /* COUNT SUBSCRIPTS */
/*  <EXPRESSION> ::= <SIMPLE EXPRESSION>    */
 IF I19 = 1 & I18 ¬= 3 THEN
 DO;CALL EMITO(IM2);I1=FIXV(LEFT);
          I20=SYLCNT;CALL EMIT2(0);
   CALL EMITO(BRS);I2=SHR(SYLCNT,2);CODE(I2)=CODEW;
   I19=0;
 END;
/*  <EXPRESSION> ::= <CASE CLAUSE> <EXPRESSION LIST> <EXPRESSION> )    */
;
/*  <EXPRESSION> ::= <IF CLAUSE> <EXPRESSION> ELSE <EXPRESSION>    */
      DO;I1=SHR(SYLCNT,2);CODE(I1)=CODEW;CALL STUFFBACK(I20,SYLCNT);
        CODEW=CODE(I1);
      END;
/*  <SIMPLE EXPRESSION> ::= <TERM>    */
;
/*  <SIMPLE EXPRESSION> ::= + <TERM>    */
;
/*  <SIMPLE EXPRESSION> ::= - <TERM>    */
   CALL EMITO(NEG);
/*  <SIMPLE EXPRESSION> ::= <SIMPLE EXPRESSION> + <TERM>    */
   CALL EMITO(ADD);
/*  <SIMPLE EXPRESSION> ::= <SIMPLE EXPRESSION> - <TERM>    */
   CALL EMITO(MIN);
/*  <TERM> ::= <TERM> * <FACTOR>    */
   CALL EMITO(MUL);
/*  <TERM> ::= <TERM> DIV <FACTOR>    */
   CALL EMITO(DIV);
/*  <TERM> ::= <FACTOR>    */
;
/*  <TERM> ::= <TERM> REM <FACTOR>    */
   CALL EMITO(REM);
/*  <FACTOR> ::= <FACTOR> ** <PRIMARY1>    */
          CALL EMITO(EXP);
/*  <FACTOR> ::= <PRIMARY>    */
;
/*  <PRIMARY1> ::= <NUMBER>    */
    GO TO NUMBERS
;
/*  <PRIMARY1> ::= <VARIABLE>    */
    GO TO VARIABLES
;
/*  <PRIMARY> ::= <NUMBER>    */
    NUMBERS:
```

```
      DO; I1=FIXV(RIGHT);
      IF I1 < 65536 THEN CALL EMITLIT(I1); ELSE
          DO; I2=GETSTORAGE(1); DATA(MAXDATA-I2)=I1;
          CALL EMITLIT(I2); CALL EMITO(INX); CALL EMITO(LOD);
          END;
      END;
/*  <PRIMARY> ::= <EXPRESSION1> )     */
;
/*  <PRIMARY> ::= <VARIABLE>     */
    VARIABLES:
          DO; I1=FIXV(LEFT);
          IF I1>=0 THEN CALL EMITO(LOD); /* SUBSCRIPTED OR SIMPLE */
          ELSE /* MAY BE A SIMPLE FUNCTION CALL (NO PARMS) */
              DO;I2=TYPE(I1);
              DO;
              I3=LOCATION(I1);
                  /* MUST BE BUILT-IN-FUNCTION OR USER DEFINED */
                      DO;
                      IF I2>PROC THEN /* INTRINSIC */
                          DO; CALL EMITLIT(I3&"7FFF"); CALL EMITO(BIF);
                          END; ELSE /* USER DEFINED FUNCTION */
                          DO; CALL EMITLIT(SHR(I3,12)&"7FFF");
                          CALL EMITO(PRO);
                          END;
                      END;
              END;
              END;
          END;
/*  <PRIMARY> ::= ABS <PRIMARY>     */
;
/*  <PRIMARY> ::= <FUNCTION PROCEDURE CALL>     */
      I19=0;
/*  <EXPRESSION1> ::= ( <EXPRESSION>     */
;
/*  <FUNCTION PROCEDURE CALL> ::= <FUNCTION CALL BODY> <IDENTIFIER> )     */
      DO; I1=FIXV(LEFT)+1; I2=SHR(I1,16); I1=I1&"FFFF";
      /* I1 IS NUMBER OF PARAMS, I2 IS SYMBOL TABLE LOCATION */
         CALL EMITLIT(LOCATION(LOOKUP(VAR(LEFT+1))));CALL EMITO(LOD);
          I3,FIXV(LEFT)=TYPE(I2); I4=I2; I2=LOCATION(I2);
      IF I3>=IFUNC & I3 <=IPROC THEN /* INTRINSIC FUNC/ PROC */
          DO; CALL EMITLIT(I2&"7FFF"); CALL EMITO(BIF);
          END; ELSE
          DO; /* USER-DEFINED FUNCTION OR PROCEDURE */
          CALL EMITLIT(SHR(I2,12)&"7FFF"); CALL EMITO(PRO);
          END;
      END;
/*  <FUNCTION PROCEDURE CALL> ::= <FUNCTION IDENTIFIER>     */
    GO TO VARIABLES;
```

```
 /*  <FUNCTION CALL BODY> ::= <FUNCTION IDENTIFIER> (    */
        DO; I1=LOOKUP(VAR(LEFT));
            I2=TYPE(I1);
             IF (I2>=FUNC)&(I2<=IPROC) THEN FIXV(LEFT)=SHL(I1,16); ELSE
                FIXV(LEFT)=SHL(I1,16);
        END;
 /*  <FUNCTION CALL BODY> ::= <FUNCTION CALL BODY> <IDENTIFIER> ,    */
  DO;
            CALL EMITLIT(LOCATION(LOOKUP(VAR(LEFT+1))));CALL EMITO(LOD);
  FIXV(LEFT)=FIXV(LEFT)+1;
  END;
 /*  <READ STATEMENT> ::= <READ HEAD> <VARIABLE> )    */
       READVAR:
        DO; CALL EMITO(RDV); CALL EMITO(STD);
        END;
 /*  <READ HEAD> ::= READ (    */
 ;
 /*  <READ HEAD> ::= READON (    */
 ;
 /*  <READ HEAD> ::= <READ HEAD> <VARIABLE> ,    */
    GO TO READVAR;
 /*  <WRITE STATEMENT> ::= <WRITE HEAD> <EXPRESSION> )    */
    WRITEVAR: CALL EMITO(WRV);
 /*  <WRITE STATEMENT> ::= <WRITE HEAD> <STRING> )    */
        DO; I2=LEFT+1; WRITESTR:
        I1=STORESTRING(VAR(I2)); CALL EMITLIT(I1);
        CALL EMITO(INX); CALL EMITO(WRS);
        END;
 /*  <WRITE HEAD> ::= WRITE (    */
        CALL EMITO(DMP);
 /*  <WRITE HEAD> ::= WRITEON (    */
 ;
 /*  <WRITE HEAD> ::= <WRITE HEAD> <EXPRESSION> ,    */
    GO TO WRITEVAR;
 /*  <WRITE HEAD> ::= <WRITE HEAD> <STRING> ,    */
        DO; I2=LEFT+1; GO TO WRITESTR;
        END;
  END;
  RETURN REDUCTION;
END SYNTHESIZE;


  /*              SYNTACTIC PARSING FUNCTIONS                    */

RIGHT_CONFLICT:
```

```
   PROCEDURE (LEFT) BIT(1);
      DECLARE LEFT FIXED;
      RETURN ("C0" & SHL(BYTE(C1(LEFT), SHR(TOKEN,2)), SHL(TOKEN,1)
      /*  THIS PROCEDURE IS TRUE IF TOKEN IS ¬ A LEGAL RIGHT CONTEXT OF LEFT*/
         & "06")) = 0;
   END RIGHT_CONFLICT;

RECOVER:
   PROCEDURE;
      IF ¬FAILSOFT THEN CALL SCAN; FAILSOFT=TRUE;
        DO WHILE FAILSOFT;
        IF SP=1 THEN
            DO; SP=SP+1; PARSE_STACK(SP)=BEGINV;
            SCANSEMI:
                DO WHILE TOKEN¬=SEMI; CALL SCAN;
                END;
            CALL SCAN; FAILSOFT=FALSE;
            END; ELSE
        IF PARSE_STACK(SP)=SEMI THEN GO TO SCANSEMI; ELSE
        IF PARSE_STACK(SP)=ENDV THEN
            DO; DO WHILE (TOKEN¬=SEMI)|(TOKEN¬=ENDV); CALL SCAN;
                END; FAILSOFT=FALSE;
            END; ELSE SP=SP-1;
        END;
      OUTPUT = 'RESUME:' || SUBSTR(POINTER, TEXT_LIMIT-CP+MARGIN_CHOP+7);
   END RECOVER;

STACKING:
   PROCEDURE BIT(1);  /* STACKING DECISION FUNCTION */
      DO FOREVER;     /* UNTIL RETURN  */
       DO CASE SHR(BYTE(C1(PARSE_STACK(SP)),SHR(TOKEN,2)),SHL(3-TOKEN,1)&6)&3;
            /*  CASE 0  */
            DO;     /* ILLEGAL SYMBOL PAIR  */
               CALL ERROR('ILLEGAL SYMBOL PAIR: ' || V(PARSE_STACK(SP)) || X1 ||
                  V(TOKEN), 1);
               CALL STACK_DUMP;
               CALL RECOVER;
            END;
            /*  CASE 1  */
            RETURN TRUE;        /*  STACK TOKEN  */
            /*  CASE 2  */
            RETURN FALSE;       /* DON'T STACK IT YET  */
            /*  CASE 3  */
            DO;         /* MUST CHECK TRIPLES  */
               J = SHL(PARSE_STACK(SP-1), 16) + SHL(PARSE_STACK(SP), 8) + TOKEN;
               I = -1;  K = NC1TRIPLES + 1;  /* BINARY SEARCH OF TRIPLES  */
               DO WHILE I + 1 < K;
                  L = SHR(I+K, 1);
```

```
                        IF C1TRIPLES(L) > J THEN K = L;
                        ELSE IF C1TRIPLES(L) < J THEN I = L;
                        ELSE RETURN TRUE;  /* IT IS A VALID TRIPLE  */
                     END;
                     RETURN FALSE;
                  END;
               END;    /* OF DO CASE  */
            END;    /*  OF DO FOREVER */
         END STACKING;

      PR_OK:
         PROCEDURE(PRD) BIT(1);
            /* DECISION PROCEDURE FOR CONTEXT CHECK OF EQUAL OR IMBEDDED RIGHT PARTS*/
            DECLARE (H, I, J, PRD) FIXED;
            DO CASE CONTEXT_CASE(PRD);

               /*  CASE 0 -- NO CHECK REQUIRED  */
               RETURN TRUE;
               /*  CASE 1 -- RIGHT CONTEXT CHECK  */
               RETURN ¬ RIGHT_CONFLICT (HDTB(PRD));
               /*  CASE 2 -- LEFT CONTEXT CHECK  */
               DO;
                  H = HDTB(PRD) - NT;
                  I = PARSE_STACK(SP - PRLENGTH(PRD));
                  DO J = LEFT_INDEX(H-1) TO LEFT_INDEX(H) - 1;
                     IF LEFT_CONTEXT(J) = I THEN RETURN TRUE;
                  END;
                  RETURN FALSE;
               END;
               /*  CASE 3 -- CHECK TRIPLES  */
               DO;
                  H = HDTB(PRD) - NT;
                  I = SHL(PARSE_STACK(SP - PRLENGTH(PRD)), 8) + TOKEN;
                  DO J = TRIPLE_INDEX(H-1) TO TRIPLE_INDEX(H) - 1;
                     IF CONTEXT_TRIPLE(J) = I THEN RETURN TRUE;
                  END;
                  RETURN FALSE;
               END;
            END;   /* OF DO CASE  */
         END PR_OK;

      REDUCE:
         PROCEDURE;
            DECLARE (I, J, PRD) FIXED;
            /* PACK STACK TOP INTO ONE WORD */
            DO I = SP - 4 TO SP - 1;
               J = SHL(J, 8) + PARSE_STACK(I);
            END;
```

```
      DO PRD = PR_INDEX(PARSE_STACK(SP)-1) TO PR_INDEX(PARSE_STACK(SP)) - 1;
         IF (PRMASK(PRLENGTH(PRD)) & J) = PRTB(PRD) THEN
            IF PR_OK(PRD) THEN
            DO;  /* AN ALLOWED REDUCTION */
               MP = SP - PRLENGTH(PRD) + 1; MPP1 = MP + 1;
               I=SYNTHESIZE(PRDTB(PRD),HDTB(PRD));
               SP=MP; PARSE_STACK(SP)=I;
               RETURN;
            END;
      END;
      /* LOOK UP HAS FAILED,ERROR CONDITION */
      CALL ERROR('NO PRODUCTION IS APPLICABLE',1);
      CALL STACK_DUMP;
      FAILSOFT = FALSE;
      CALL RECOVER;
   END REDUCE;

COMPILATION_LOOP:
   PROCEDURE;

      COMPILING = TRUE;
      DO WHILE COMPILING;      /* ONCE AROUND FOR EACH PRODUCTION (REDUCTION)  */
         DO WHILE STACKING;
            SP = SP + 1;
            IF SP = STACKSIZE THEN
               DO;
                  CALL ERROR ('STACK OVERFLOW *** CHECKING ABORTED ***', 2);
                  RETURN;    /* THUS ABORTING CHECKING */
               END;
            PARSE_STACK(SP) = TOKEN;
            VAR(SP) = BCD;
              IF BCD='THEN' THEN    I19=1;
            IF BCD='WHILE' THEN I14=SYLCNT;
            FIXV(SP) = NUMBER_VALUE;
            CALL SCAN;
       IF PARSE_STACK(SP)=30 THEN CALL STUFFBACK(I13,SYLCNT);
         END;

         CALL REDUCE;
      END;      /* OF DO WHILE COMPILING  */
   END COMPILATION_LOOP;



PRINT_SUMMARY:
   PROCEDURE;
      DECLARE I FIXED;
```

```
   CALL PRINT_DATE_AND_TIME ('END OF CHECKING ', DATE, TIME);
   OUTPUT = '';
   OUTPUT = CARD_COUNT || ' CARDS WERE CHECKED.';
   IF ERROR_COUNT = 0 THEN OUTPUT = 'NO ERRORS WERE DETECTED.';
   ELSE IF ERROR_COUNT > 1 THEN
      OUTPUT = ERROR_COUNT || ' ERRORS (' || SEVERE_ERRORS
         || ' SEVERE) WERE DETECTED.';
   ELSE IF SEVERE_ERRORS = 1 THEN OUTPUT = 'ONE SEVERE ERROR WAS DETECTED.';
      ELSE OUTPUT = 'ONE ERROR WAS DETECTED.';
   IF PREVIOUS_ERROR > 0 THEN
      OUTPUT = 'THE LAST DETECTED ERROR WAS ON LINE ' || PREVIOUS_ERROR
         || PERIOD;
   IF CONTROL(BYTE('D')) THEN CALL DUMPIT;
   DOUBLE_SPACE;
   CLOCK(3) = TIME;
   DO I = 1 TO 3;    /* WATCH OUT FOR MIDNIGHT */
      IF CLOCK(I) < CLOCK(I-1) THEN CLOCK(I) = CLOCK(I) +  8640000;
   END;
   CALL PRINT_TIME ('TOTAL TIME IN CHECKER    ', CLOCK(3) - CLOCK(0));
   CALL PRINT_TIME ('SET UP TIME              ', CLOCK(1) - CLOCK(0));
   CALL PRINT_TIME ('ACTUAL CHECKING TIME     ', CLOCK(2) - CLOCK(1));
   CALL PRINT_TIME ('CLEAN-UP TIME AT END     ', CLOCK(3) - CLOCK(2));
   IF CLOCK(2) > CLOCK(1) THEN   /* WATCH OUT FOR CLOCK BEING OFF */
   OUTPUT = 'CHECKING RATE: ' || 6000*CARD_COUNT/(CLOCK(2)-CLOCK(1))
      || ' CARDS PER MINUTE.';
END PRINT_SUMMARY;

 DUMPFILE: PROCEDURE;
 PRTCNT=PRTMAX;
 /* #PRT, #DATA, #CODE, #STACK ARE PLACED INTO FILE FIRST */
 I1=DATALENGTH;
      DO WHILE I1>3; /* MOVE EVERYTHING UP 4 SPACES */
      DATA(I1)=DATA(I1-4);
      I1=I1-1;
      END;
 SYLCNT=(SYLCNT+3)/4-1;
 DATA(0)=PRTCNT; DATA(1)=DATACNT; DATA(2)=SYLCNT; DATA(3)=EXSTACK;
 I1=4+PRTCNT; I2=DATALENGTH;
      DO WHILE I1 < I2;
      I3=DATA(I1);DATA(I1)=DATA(I2); DATA(I2)=I3; I1=I1+1; I2=I2-1;
      END;
 I1=4+PRTCNT+DATACNT;
 /* NOW PLACE CODE INTO FILE 1 */
 I4=0; I2=SYLCNT-1;
      DO I3=0 TO I2;
      IF I1=DATALENGTH THEN
          DO; FILE(1,I4)=DATA; I4=I4+1; I1=0;
          END;
```

```
            DATA(I1)=CODE(I3); I1=I1+1;
            END;
            DO I3=I1 TO DATALENGTH;  DATA(I3)=0;
            END;
        FILE(1,I4)=DATA;
        OUTPUT='CODE FILE WRITTEN';
            RETURN;
        END DUMPFILE;

MAIN_PROCEDURE:
    PROCEDURE;
        CLOCK(0) = TIME;  /* KEEP TRACK OF TIME IN EXECUTION */
        CALL INITIALIZATION;
        CLOCK(1) = TIME;
        CALL COMPILATION_LOOP;
      OUTPUT='PRT='||PRTMAX||',DATA='||DATACNT||',CODE='||SHR(SYLCNT,2)
      ||'(WORDS).';
        IF SEVERE_ERRORS=0 THEN CALL DUMPFILE;
        CLOCK(2) = TIME;
        /* CLOCK(3) GETS SET IN PRINT_SUMMARY */
        CALL PRINT_SUMMARY;
    END MAIN_PROCEDURE;

CALL MAIN_PROCEDURE;

EOF EOF EOF
```

APPENDIX C

SAMPLE PROGRAM

```
                ALGOL-W(SUBSET) COMPILATION

TODAY IS MAY 21, 1972.  CLOCK TIME = 10:39:55.10.

CARD |BL| SYL |
   1 | 0|   0 |BEGIN COMMENT ALGOL-W(SUBSET) PROGRAM TO COMPUTE PERMUTATIONS OF
   2 | 0|   0 |                SETS OF NUMBERS;
   3 | 0|   0 |  INTEGER N;
   4 | 1|   0 |INTEGER PROCEDURE FACTORIAL(INTEGER VALUE N);
   5 | 2|  10 |  BEGIN INTEGER I;I:=N-1;
   6 | 3|  16 |  IF N = 0 THEN 1 ELSE N*FACTORIAL(I)
   7 | 3|  31 |  END;
   8 | 1|  46 |  READ(N);
   9 | 1|  49 |  WHILE N ¬= 9999 DO
  10 | 1|  55 |  BEGIN INTEGER ARRAY SET(1::N);
  11 | 2|  68 |  PROCEDURE WRITEOUT;
  12 | 3|  78 |    BEGIN WRITE(" ");
  13 | 4|  82 |      FOR I:=1 STEP 1 UNTIL N DO
  14 | 4|  86 |        WRITEON(SET(I));
  15 | 4| 127 |    END;
  16 | 2| 134 |  PROCEDURE EXCHANGE(INTEGER VALUE I,J);
  17 | 3| 144 |    BEGIN INTEGER TEMP; TEMP:=SET(I);SET(I):=SET(J);
  18 | 4| 164 |      SET(J):=TEMP
  19 | 4| 169 |    END;
  20 | 2| 179 |  PROCEDURE PERMUTE(INTEGER VALUE K);
  21 | 3| 189 |    IF K = N THEN WRITEOUT ELSE
  22 | 3| 201 |    BEGIN
  23 | 3| 201 |      FOR I:=K STEP 1 UNTIL N DO
  24 | 4| 206 |        BEGIN EXCHANGE(I,K);PERMUTE(K+1);
  25 | 5| 250 |          EXCHANGE(K,I)
  26 | 5| 254 |        END;
  27 | 4| 264 |    END;
  28 | 2| 277 |  PROCEDURE READER;
  29 | 3| 287 |    BEGIN
  30 | 3| 287 |      FOR I:=1 STEP 1 UNTIL N DO READ(SET(I));
  31 | 4| 332 |        WRITEOUT;
  32 | 4| 335 |    END;
  33 | 2| 342 |  WRITE(" ");WRITE("THERE ARE",FACTORIAL(N));
  34 | 2| 355 |  WRITEON("PERMUTATIONS OF THE FOLLOWING SET OF NUMBERS");
  35 | 2| 358 |  READER;WRITE(" ");WRITE(" ");
  36 | 2| 371 |  PERMUTE(1);WRITE(" ");READ(N);
  37 | 2| 383 |  END;
  38 | 1| 390 |  WRITE("END OF RUN")
  39 | 1| 391 |END.
  40 | 0| 394 |EOF
PRT=23,DATA=23,CODE=99(WORDS).
CODE FILE WRITTEN
END OF CHECKING MAY 21, 1972.  CLOCK TIME = 10:39:57.95.

40 CARDS WERE CHECKED.
NO ERRORS WERE DETECTED.


TOTAL TIME IN CHECKER     0:0:3.21.
SET UP TIME               0:0:0.41.
ACTUAL CHECKING TIME      0:0:2.57.
CLEAN-UP TIME AT END      0:0:0.23.
CHECKING RATE: 933 CARDS PER MINUTE.

PRT=23, DATA=23, CODE=99
MEMORY INITIALIZED, STARTING AT 345, ENDING AT 8191
```

```
THERE ARE       24              PERMUTATIONS OF THE FOLLOWING SET OF NUMBERS
1               2               3               4

                1               2               3               4
                1               2               4               3
                1               3               2               4
                1               3               4               2
                1               4               3               2
                1               4               2               3
                2               1               3               4
                2               1               4               3
                2               3               1               4
                2               3               4               1
                2               4               3               1
                2               4               1               3
                3               2               1               4
                3               2               4               1
                3               1               2               4
                3               1               4               2
                3               4               1               2
                3               4               2               1
                4               2               3               1
                4               2               1               3
                4               3               2               1
                4               3               1               2
                4               1               3               2
                4               1               2               3

THERE ARE       6               PERMUTATIONS OF THE FOLLOWING SET OF NUMBERS
100             25              389

                100             25              389
                100             389             25
                25              100             389
                25              389             100
                389             25              100
                389             100             25

END OF RUN
```

LIST OF REFERENCES

1. Kildall, Gary A., ALGOL-E: An Experimental Approach to the Study of Programming Languages, Naval Postgraduate School, 1972.

2. Bauer, Henry R., and others, ALGOL-W (REVISED), Stanford University, 1969.

3. McKeeman, William M., Horning, James J., and Wortman, David B., A Compiler Generator, Prentice-Hall, Inc., 1970.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	LtJG Alan B. Roberts, USNR Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	Assistant Professor Gary A. Kildall Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
5.	Lt. Carl C. Brewer, USN 245 East Avenue Minoa, New York 13116	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

ALGOL-W (Subset) Compiler

4. DESCRIPTIVE NOTES *(Type of report and, inclusive dates)*

Master's Thesis; June 1972

5. AUTHOR(S) *(First name, middle initial, last name)*

Carl Clifford Brewer

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1972	64	3

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

This paper describes the ALGOL-W (subset) compiler and language. The primary objective of this project was to develop a compiler that would accept a modified integer version of ALGOL-W that could be used as a programming system to aid in the study of the compilation and execution of ALGOL-like languages. Included is a brief discussion of the ALGOL-E machine options that complement the compiler's capabilities. This compiler was written in the XPL language.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31406

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Compiler						
ALGOL-W						
ALGOL-E						

Thesis
B8048 Brewer
c.1 ALGOL-W (subset) com-
piler.
135267